VISIT…

Programmer to Programmer™

Professional VMware® Server

Eric Hammersley

wrox

Updates, source code, and Wrox technical support at www.wrox.com

Professional
VMware® Server

Eric Hammersley

Wiley Publishing, Inc.

Professional
VMware® Server

Eric Hammersley

Wiley Publishing, Inc.

Professional VMware® Server

Published by
Wiley Publishing, Inc.
10475 Crosspoint Boulevard
Indianapolis, IN 46256
www.wiley.com

Copyright © 2007 by Wiley Publishing, Inc., Indianapolis, Indiana

Published simultaneously in Canada

ISBN-13: 978-0-470-07988-1
ISBN-10: 0-470-07988-6

Manufactured in the United States of America

10 9 8 7 6 5 4 3 2 1

1B/SR/RR/QW/IN

Library of Congress Cataloging-in-Publication Data:

Hammersley, Eric, 1974-
Professional VMware server / Eric Hammersley.
p. cm.
Includes index.
ISBN-13: 978-0-470-07988-1 (paper/website)
ISBN-10: 0-470-07988-6 (paper/website)
1. VMware. 2. Operating systems (Computers) 3. Virtual computer systems. I. Title.
QA76.76.O63H344637 2007
005.4'3--dc22

2006033472

No part of this publication may be reproduced, stored in a retrieval system or transmitted in any form or by any means, electronic, mechanical, photocopying, recording, scanning or otherwise, except as permitted under Sections 107 or 108 of the 1976 United States Copyright Act, without either the prior written permission of the Publisher, or authorization through payment of the appropriate per-copy fee to the Copyright Clearance Center, 222 Rosewood Drive, Danvers, MA 01923, (978) 750-8400, fax (978) 646-8600. Requests to the Publisher for permission should be addressed to the Legal Department, Wiley Publishing, Inc., 10475 Crosspoint Blvd., Indianapolis, IN 46256, (317) 572-3447, fax (317) 572-4355, or online at http://www.wiley.com/go/permissions.

LIMIT OF LIABILITY/DISCLAIMER OF WARRANTY: THE PUBLISHER AND THE AUTHOR MAKE NO REPRESENTATIONS OR WARRANTIES WITH RESPECT TO THE ACCURACY OR COMPLETENESS OF THE CONTENTS OF THIS WORK AND SPECIFICALLY DISCLAIM ALL WARRANTIES, INCLUDING WITHOUT LIMITATION WARRANTIES OF FITNESS FOR A PARTICULAR PURPOSE. NO WARRANTY MAY BE CREATED OR EXTENDED BY SALES OR PROMOTIONAL MATERIALS. THE ADVICE AND STRATEGIES CONTAINED HEREIN MAY NOT BE SUITABLE FOR EVERY SITUATION. THIS WORK IS SOLD WITH THE UNDERSTANDING THAT THE PUBLISHER IS NOT ENGAGED IN RENDERING LEGAL, ACCOUNTING, OR OTHER PROFESSIONAL SERVICES. IF PROFESSIONAL ASSISTANCE IS REQUIRED, THE SERVICES OF A COMPETENT PROFESSIONAL PERSON SHOULD BE SOUGHT. NEITHER THE PUBLISHER NOR THE AUTHOR SHALL BE LIABLE FOR DAMAGES ARISING HEREFROM. THE FACT THAT AN ORGANIZATION OR WEBSITE IS REFERRED TO IN THIS WORK AS A CITATION AND/OR A POTENTIAL SOURCE OF FURTHER INFORMATION DOES NOT MEAN THAT THE AUTHOR OR THE PUBLISHER ENDORSES THE INFORMATION THE ORGANIZATION OR WEBSITE MAY PROVIDE OR RECOMMENDATIONS IT MAY MAKE. FURTHER, READERS SHOULD BE AWARE THAT INTERNET WEBSITES LISTED IN THIS WORK MAY HAVE CHANGED OR DISAPPEARED BETWEEN WHEN THIS WORK WAS WRITTEN AND WHEN IT IS READ.

For general information on our other products and services please contact our Customer Care Department within the United States at (800) 762-2974, outside the United States at (317) 572-3993 or fax (317) 572-4002.

Trademarks: Wiley, the Wiley logo, Wrox, the Wrox logo, Programmer to Programmer, and related trade dress are trademarks or registered trademarks of John Wiley & Sons, Inc. and/or its affiliates, in the United States and other countries, and may not be used without written permission. VMware is a registered trademark of VMware, Inc. All other trademarks are the property of their respective owners. Wiley Publishing, Inc., is not associated with any product or vendor mentioned in this book.

Wiley also publishes its books in a variety of electronic formats. Some content that appears in print may not be available in electronic books.

To Sandy, my wonderful wife, and our two children, Chandler and Renae. Without you this book would never have been possible. Your support and encouragement helped make this book what it is today. I love you all dearly.

About the Author

Eric Hammersley has been a technician, systems engineer, project manager, and developer in the IT industry for more than 15 years. He began his career in networking and support in the early '90s installing and supporting hundreds of networks during that time. He then moved into the government sector joining the United States Navy where he remained on active duty through the turn of the century. Although he specialized in shipboard weapons and radar systems, his focus did not shy away from the IT field. Taking charge of the shipboard computer systems he later transitioned into the Integrated Shipboard Network System (ISNS) project—a Navy initiative in the late '90s to integrate the data systems of units afloat to help improve Command, Control, and Communication across the fleet. It was during the ISNS project, just after VMware first released its Workstation product, that he realized the potential of software virtualization. Having to cram racks of servers into an already overcrowded shipboard environment was a wake-up call. A better way had to exist, and VMware was the frontrunner in that technology. From that point forward he became an avid proponent of software virtualization. Throughout the evolution of the VMware product line he has continued his quest to increase the awareness and use of software virtualization by the software developer. He is currently the CIO and Lead Software Architect for The SASSI Institute and resides in Bedford, Indiana. You can contact him at `eric@vmcoder.com`—a site that focuses on the use of virtualization throughout the software development cycle.

Acknowledgments

None of this could have been possible without the wonderful team at Wrox. I would like to thank Chris Webb, who took a chance and gave me a shot at this. I am forever grateful for the opportunity. A special thanks to Kevin Kent. Without your amazing skills this book would be little more than thoughts on a dinner napkin. You are truly a pleasure to work with. To everyone else at Wrox a heartfelt thanks. Each and every one of you involved helped make this book a success.

A special thanks to Scott Lowe. Your technical expertise and point of view were important parts of this process.

Again, I would like to thank my wonderful family. I would be lost without you. You sacrificed a great deal over the past several months to make this book a reality. Without your love and support this would not have been possible.

I would also like to thank my parents, Larry and Sue. You have always been there for me. Without hesitation you stepped in and helped over the past several months. I love you both.

Credits

Executive Editor
Chris Webb

Senior Development Editor
Kevin Kent

Technical Editor
Scott Lowe

Production Editor
Eric Charbonneau

Copy Editor
Kim Cofer

Editorial Manager
Mary Beth Wakefield

Production Manager
Tim Tate

Vice President and Executive Group Publisher
Richard Swadley

Vice President and Executive Publisher
Joseph B. Wikert

Graphics and Production Specialists
Carrie A. Foster
Barry Offringa
Rashell Smith
Alicia B. South

Quality Control Technicians
John Greenough
Christy Pingleton

Project Coordinator
Erin Smith

Proofreading and Indexing
Techbooks

Anniversary Logo Design
Richard Pacifico

Contents

Acknowledgments iii
Introduction xiii
Who This Book Is For xiii
What This Book Covers xiv
How This Book Is Structured xiv
What You Need to Use This Book xvi
Conventions xvi
Source Code xvii
Errata xvii
p2p.wrox.com xvii

Chapter 1: VMware Server: A Brief Introduction 1

What Is Virtualization? 1
Software Virtualization 2
Hosted vs. Hypervisor Architecture 3
Terminology, Host vs. Guest 6
So What Is VMware Server? 6
Rapid Deployment 6
Multiple Platforms Available on a Single Physical Machine 7
The Ability to Move Virtual Machines from One Server to Another 7
Increased Utilization of Your Hardware 8
Life without VMware Server 8
Summary 9

Chapter 2: Installing Your VMware Server 11

Host Hardware Requirements 11
Processor 12
Memory 12
Hard Disk 13
Network Controller 14
Host Operating System Requirements 14
Windows 15
Linux 15

Remote Client Requirements **17**
Hardware 17
Operating System 18
Installing VMware Server **18**
Installing VMware Server on Windows 19
Installing VMware Server on Linux 24
Install Overview 34
Installing the VMware Server Management Interface on Linux **34**
Installing the Remote Client **38**
Installing the Remote Client on Windows 38
Installing the Remote Client on Linux 43
Troubleshooting the Install **48**
Having to Compile Your Own Kernel Modules 48
Verifying Port 902 Is Free 50
Installing xinetd 50
The Case of the Missing Library: libdb.so.3 51
Launching the Server Console Manually 52
Enabling the MUI and Console Ports in Your Firewall 54
Summary **56**

Chapter 3: VMware Configuration Nuts and Bolts **57**

Getting to Know the Server Console **57**
Host Settings and Console Preferences **62**
Host Settings 62
Console Preferences 66
Virtual Machine Settings **72**
Virtual Hardware Specifications 73
Virtual Machine Options 75
Virtual Networking **84**
Automatic Bridging 86
Host Virtual Network Mapping 87
Host Virtual Adapters 91
DHCP and NAT 92
Virtual Machine Security **92**
Options for User Access Levels 92
Configuring User Access Levels 93
Summary **96**

Chapter 4: Creating Your First Base Image **97**

What Is a Base Image? **97**
Creating Your First Virtual Machine **98**

Installing a Guest Operating System **109**
Troubleshooting Guest Operating System Installation 112
Installing VMware Tools on Windows Guests 113
Installing VMware Tools on Linux Guests 113
Freezing Your Windows Guest with Sysprep 116
Summary **123**

Chapter 5: Where Does This Fit into Your Development Environment? **125**

Virtual Machines for Software Development **126**
Virtual Machines for Software Testing **128**
Platform Testing with Snapshots 129
Platform Testing with Persistent and Nonpersistent Disks 130
Virtual Machines for Post-Release Maintenance **131**
Creating Base Images for Your Development Environment **131**
Planning 132
Collaborating 133
Creating 133
Testing 134
Deploying 135
Summary **136**

Chapter 6: Bringing It All Together **137**

Organizing Your Image Library **137**
Proper Image Naming 137
Archiving Your Images 140
Preparing Base Images for Use **140**
Recycling Your Testing Images **143**
Creating an ISO Image Library **144**
Creating an ISO Image 145
Using an ISO Image 146
Summary **148**

Chapter 7: Introduction to the Scripting APIs **151**

What Can You Automate? **152**
Supported Platforms **153**
VmPerl 153
VmCOM 153
Programming API 154
VmPerl vs. VmCOM vs. Vix **154**
The vmware-cmd Utility **156**

The vmrun Utility **157**
The VMware SDK **157**
Summary **157**

Chapter 8: Using VmCOM **159**

A Quick OOP Primer **159**
Windows Scripting Host **160**
Objects **161**
VmConnectParams 161
VmServerCtl 162
VmCtl 164
VmCollection 171
VmQuestion 173
Enumerations **177**
VmExecutionState 177
VmPowerOpMode 178
VmProdInfoType 179
VmProduct 179
VmPlatform 179
Summary **180**

Chapter 9: Using VmPerl **183**

Cross-Platform Perl Scripts **183**
Modules **185**
VMware::VmPerl::ConnectParams 185
VMware::VmPerl::Server 187
VMware::VmPerl::VM 191
VMware::VmPerl::Question 199
Constants **204**
VM_EXECUTION_STATE 204
VM_POWEROP_MODE 204
Infotype 205
VM_PRODINFO_PRODUCT 206
VM_PRODINFO_PLATFORM 206
Summary **207**

Chapter 10: Using the Programming API **209**

Compiling Against Vix with Visual C++ **209**
Setting Up the Project 210
Including the include Statement 215

Pointing the Project to the Header Files 216
Pointing the Project to the Libraries Used 218
A Quick Vix Primer 220
Handles 220
Handle Types 222
Handle Properties 222
Error Codes 225
Asynchronous Operations and the Job Handle 226
Functions 230
Vix_ReleaseHandle 231
Vix_GetProperties 231
Vix_GetPropertyType 232
Vix_GetHandleType 233
Vix_GetErrorText 234
Vix_FreeBuffer 235
Vix_CreateTemporaryObject 236
Vix_PumpEvents 236
Host Machine Operations 237
VixHost_Connect 237
VixHost_Disconnect 239
VixHost_FindItems 240
VixHost_RegisterVM 243
VixHost_UnregisterVM 244
Virtual Machine Operations 245
VixVM_Open 245
VixVM_PowerOn 246
VixVM_PowerOff 247
VixVM_Reset 248
VixVM_Suspend 249
VixVM_Delete 250
VixVM_CreateSnapshot 251
VixVM_GetNumRootSnapshots 253
VixVM_GetRootSnapshot 254
VixVM_RemoveSnapshot 255
VixVm_RevertToSnapshot 256
VixVM_InstallTools 258
VixVM_WaitForToolsInGuest 259
VixVM_UpgradeVirtualHardware 261
VixVM_LoginInGuest 262
VixVM_RunProgramInGuest 263
VixVM_CopyFileFromHostToGuest 265
VixVM_CopyFileFromGuestToHost 266

Job Handle Operations **268**
VixJob_Wait 268
VixJob_GetError 268
VixJob_CheckCompletion 269
Using Vix — A Few Brief Examples **269**
Summary **275**

Chapter 11: Integrating with Your Existing Environment **277**

Continuous Integration **278**
VMware Tools Scripting **279**
Scripting VMware Tools with VBScript 279
Scripting VMware Tools with C# and VB.NET 283
CruiseControl.NET **285**
Basic Configuration 285
VMware Server Integration 299
Visual Studio Team System **312**
Creating a Team Project 312
Adding a Development Project to the Team Project 318
Configuring a Team Build 322
Testing the Build 328
VMware Server Integration 330
VMware Server Integration under Linux — A Brief Look **349**
Summary **349**

Chapter 12: A Real-World Scenario **351**

Defining Your Environment **351**
Target Platform 352
Supported Platforms 352
Tools 352
Source and Process Control 353
Defining Your Goals **353**
Setting Up Your Development Images **354**
Setting up the Virtual Development Machine 355
Deleting the Virtual Development Machine Floppy Drive 365
Renaming the Virtual Development Machine Configuration File 365
Setting up the Virtual Development Machine Software for Deployment 367
Setting Up Your Testing Images **374**
Installing the VMware Tools Script to Retrieve the Guest IP Address 375
Configuring Your Testing Images as Nonpersistent 378

Configuring Build Integration **380**
Project Setup 381
VMware Server Integration 394
The Final Scripts 403
Summary **408**

Appendix A: VmCOM Error Codes **409**

Appendix B: VmPerl Error Codes **413**

Appendix C: Vix Error Codes **417**

Index **423**

Introduction

Software virtualization is not a new concept; however, it is only recently that it has begun to gain momentum by the masses. This movement can be attributed in large part to the innovations of VMware, a company that has specialized in virtualization technology since the late '90s.

Early on, the release of VMware Workstation was used mainly by advanced users and hobbyists. This was due in large part to the lack of understanding of the technology as well as the progression of the industry as a whole. This quickly changed, however, as the technology matured into not only a more stable solution but one that the masses could easily utilize. Now it is not uncommon to find even the most basic users dual-booting their workstations, wanting to try out Linux or some other operating system, or just having a clean machine to play with. This move triggered the recognition of VMware Workstation as more than just a specialist's tool. With it, even a beginner can configure and operate a virtual machine with little or no previous experience.

This boom also gave way to new and innovative ways for greater utilization in the data center, giving companies from small to large the ability to increase their return on the investment of hardware and flourish. It doesn't stop there, however. The technology offers great advantages to the software developer and tester alike. It's such a perfect fit you'd think it was its sole purpose in life. Sadly, this seems to be the tail runner in the virtualization boom. Developers are still writing code on their office workstations and testing in rooms that contain dozens of individual PCs. Why?

This book focuses on VMware Server, a now free server-based version of VMware's popular virtualization line. Throughout this book I discuss not only the benefits of software development and testing within virtual machines, but the advantages they offer in terms of automation, stability, disaster recovery, and overall code quality.

Who This Book Is For

This book is designed to not only teach you how to use VMware Server but how to automate its function through the use of its APIs.

A few specific categories of readers stand to gain the most from this book:

- **Software Developers**—The software developer is one of the main players here. Every day they sit down either as an individual or as a team and crank out the software we all use on a daily basis. Without them, the industry wouldn't exist. This book introduces the software developer to the benefits of virtual machines and what they can offer. For the developer that already has an understanding of their benefits, this book takes them into more advanced topics, such as base images, test banks, and automation.

- **Lead Developers**—Though this title may not differ a whole lot from the software developer itself, the roles are greatly different. This book aims to provide the software development team leads out there with the information required to make the move into team development within virtual machines. It discusses development images, enforcing standards on your development team, and project management beyond the development cycle—specifically, how software development within virtual machines allows for archival and later reference of exact development environments long after a product is released.
- **Software Testing Engineers**—Some of the real meat of this book centers around the automation available to increase the exposure your applications receive during their testing phase. You will learn how to integrate VMware Server into your existing systems to offer automated builds and testing on banks of virtual machines.

What This Book Covers

This books covers VMware Server, a free server-based virtualization product from VMware. It does not, however, stop there. A serious amount of attention is also provided to the legacy programming APIs, VmPerl and VmCOM, as well as the newest API, the Programming API. This book guides you through creating a fleet of virtual machine images that can be used for development or testing purposes, as well as how to organize them. It also gives heavy focus on the automation of VMware Server and its integration into your existing process management tools like CruiseControl.Net and Visual Studio Team System.

The first few chapters encompass the basics of virtualization and its place within the development and testing environment. It then quickly moves into the APIs and provides a deep, example-rich reference to their function and purpose. The final chapters walk you through integrating VMware Server into your existing environment using CruiseControl.NET and Visual Studio Team System as examples, as well as provide a more "real-world" type example to help tighten the concept up.

Although this book is designed around VMware Server, because it is a free product there is nothing that keeps it from being equally applicable to other VMware products like ESX Server.

How This Book Is Structured

The goal is a front-to-back approach to introducing virtualization into your development process. The already familiar reader can easily skim the first chapter; however, starting right in at Chapter 2 I get down and dirty with VMware Server and start outlining just how you can set it up for your development process. The API chapters, Chapters 8, 9, and 10, offer an easy reference to the API functions and samples of their use. They should provide a handy desk reference for some time to come.

- **Chapter 1, "VMware Server: A Brief Introduction"**—This chapter introduces virtualization as a whole and provides some insight into the technology, its evolution, and its uses.
- **Chapter 2, "Installing Your VMware Server"**—This chapter walks you through installing VMware Server, its requirements, and its individual pieces. This chapter also reviews the hardware and software requirements at the time of writing as well as the virtual hardware specifications your virtual machines will utilize.

- **Chapter 3, "VMware Configuration Nuts and Bolts"**—This chapter really dives into the different settings available with the VMware Server product. From the host settings to virtual machines settings, this is an all-encompassing look into the product itself. It also discusses virtual networking and virtual machine security in a networked environment.
- **Chapter 4, "Creating Your First Base Image"**—This chapter covers the base image and its creation. A base image is essentially a virtual machine configured with an operating system installed from which you will base all other virtual machines. Topics include creating the virtual machine, installing the operating system, and freezing the install with Sysprep, a Windows-based installation tool.
- **Chapter 5, "Where Does This Fit into Your Development Environment?"**—This chapter begins the trek into virtual machines for software development and testing. Here I discuss their use in the development process as well as some good tips to follow when creating your development base images for your team. I also discuss post-release maintenance and methods to introduce virtual machine development to your team.
- **Chapter 6, "Bringing It All Together"**—This chapter brings Chapters 4 and 5 together and explains the methods in which you can organize your image libraries and prepare them for use, and organize your testing images and ISO image libraries.
- **Chapter 7, "Introduction to the Scripting APIs"**—This chapter introduces the APIs available for VMware Server. Here I discuss their supported platforms and the pros and cons of each. This chapter is a lead into the reference chapters ahead.
- **Chapter 8, "Using VmCOM"**—This is the first of three API reference chapters. Here I introduce the VmCOM API and its functionality. I review the specific objects and methods available as well as provide example code to help you leverage the VmCOM API in production.
- **Chapter 9, "Using VmPerl"**—This chapter introduces the VmPerl API. Here I discuss its use, functionality, and flexibility within the product line. Topics include a module reference complete with their corresponding objects and methods as well as example code to help you leverage it in real life.
- **Chapter 10, "Using the Programming API"**—This is a rather lengthy look into the newest API offering from VMware, the Programming API, or Vix. I set up a development environment for your scripts utilizing C++, which is currently the only language supported, and provide a primer of sorts into Vix and its methodology. The Vix API contains a great deal more functionality than the previous two API offerings and therefore is a bit more complex; however, once it matures, it will be the de facto standard in VMware automation.
- **Chapter 11, "Integrating with Your Existing Environment"**—This chapter takes on two common integration packages, CruiseControl.NET and Visual Studio Team System. Here I outline how to integrate VMware Server into those environments and automate its functionality. The main goal is to automate builds, test bank virtual machine allocation, and the pushing of test builds up to those machines.
- **Chapter 12, "A Real-World Scenario"**—This chapter brings everything together in a real-world scenario of sorts. Here I take CruiseControl.NET, a popular free continuous integration package, and integrate it into an existing development environment with specific needs and specifications. I'll show how to leverage VMware Server to provide banks of testing machines on demand all through a nightly build script.
- **Appendixes A, B, and C, "Error Codes"**—To wrap things up, the appendixes outline the various error codes by API and provide a quick reference to their use, trapping, and identification.

What You Need to Use This Book

You will need at the very least a PC with an approved operating system for VMware Server as well as the VMware Server software itself. The fact that VMware Server is free makes it a lot easier.

Chapter 2 outlines the approved server-based operating systems as well as the installation procedure for VMware Server.

The coding examples and scenarios for the VmPerl and VmCOM APIs require no additional software support. They can be compiled and run either natively by the operating system itself or by the Perl interpreter provided by VMware Server. The Programming API examples are written in Visual C++ 2005. They can, however, be compiled using the command-line compiler provided by the .NET Framework.

Where appropriate, Linux examples are provided. Two distributions are utilized in this book; however, any can be used that fall within the specifications provided for VMware Server. Specifically, I use Red Hat Enterprise Linux and Fedora Core. The versions used are pointed out in their respective sections.

The final two chapters that deal with integration require either CruiseControl.NET, a free continuous integration product, or Visual Studio Team System (VSTS). Scenarios are provided utilizing both; however, if you do not have access to VSTS, there's still a great deal of value to be had. The final chapter, the real-world scenario, is written against CruiseControl.NET. This was done to keep all the readers, regardless of their access to VSTS, in the loop.

Conventions

To help you get the most from the text and keep track of what's happening, we've used a number of conventions throughout the book.

Boxes like this one hold important, not-to-be forgotten-information that is directly relevant to the surrounding text.

Tips, hints, tricks, and asides to the current discussion are offset and placed in italics like this.

As for styles in the text:

- We *highlight* new terms and important words when we introduce them.
- We show keyboard strokes like this: Ctrl+A.
- We show file names, URLs, and code within the text like so: `persistence.properties`.
- We present code in two different ways:

```
In code examples we highlight new and important code with a gray background.

The gray highlighting is not used for code that's less important in the present
context, or has been shown before.
```

Please also note that in cases where a continuous code line has been broken due to the constraints of the margins of the book, you will see the ↩ symbol indicating that the code line continues onto the following line.

Source Code

As you work through the examples in this book, you may choose either to type in all the code manually or to use the source code files that accompany the book. All of the source code used in this book is available for download at `http://www.wrox.com`. Once at the site, simply locate the book's title (either by using the Search box or by using one of the title lists) and click the Download Code link on the book's detail page to obtain all the source code for the book.

> *Because many books have similar titles, you may find it easiest to search by ISBN; this book's ISBN is 0-470-07988-6 (changing to 978-0-470-07988-1 as the new industry-wide 13-digit ISBN numbering system is phased in by January 2007).*

Once you download the code, just decompress it with your favorite compression tool. Alternatively, you can go to the main Wrox code download page at `http://www.wrox.com/dynamic/books/download.aspx` to see the code available for this book and all other Wrox books.

Errata

We make every effort to ensure that there are no errors in the text or in the code. However, no one is perfect, and mistakes do occur. If you find an error in one of our books, like a spelling mistake or faulty piece of code, we would be very grateful for your feedback. By sending in errata you may save another reader hours of frustration and at the same time you will be helping us provide even higher quality information.

To find the errata page for this book, go to `http://www.wrox.com` and locate the title using the Search box or one of the title lists. Then, on the book details page, click the Book Errata link. On this page you can view all errata that has been submitted for this book and posted by Wrox editors. A complete book list including links to each book's errata is also available at `www.wrox.com/misc-pages/booklist.shtml`.

If you don't spot "your" error on the Book Errata page, go to `www.wrox.com/contact/techsupport.shtml` and complete the form there to send us the error you have found. We'll check the information and, if appropriate, post a message to the book's errata page and fix the problem in subsequent editions of the book.

p2p.wrox.com

For author and peer discussion, join the P2P forums at `p2p.wrox.com`. The forums are a Web-based system for you to post messages relating to Wrox books and related technologies and interact with other readers and technology users. The forums offer a subscription feature to e-mail you topics of interest of your choosing when new posts are made to the forums. Wrox authors, editors, other industry experts, and your fellow readers are present on these forums.

At `http://p2p.wrox.com` you will find a number of different forums that will help you not only as you read this book, but also as you develop your own applications. To join the forums, just follow these steps:

1. Go to `p2p.wrox.com` and click the Register link.
2. Read the terms of use and click Agree.
3. Complete the required information to join as well as any optional information you wish to provide and click Submit.
4. You will receive an e-mail with information describing how to verify your account and complete the joining process.

You can read messages in the forums without joining P2P; but in order to post your own messages, you must join.

Once you join, you can post new messages and respond to messages other users post. You can read messages at any time on the Web. If you would like to have new messages from a particular forum e-mailed to you, click the Subscribe to this Forum icon by the forum name in the forum listing.

For more information about how to use the Wrox P2P, be sure to read the P2P FAQs for answers to questions about how the forum software works as well as many common questions specific to P2P and Wrox books. To read the FAQs, click the FAQ link on any P2P page.

1

VMware Server: A Brief Introduction

VMware has been on the cutting edge of virtualization since the release of VMware Workstation in 1999. In 2001 VMware released its first of a long line of server-based products. The VMware ESX and GSX Server products have changed the way we look at our server utilization to this day. There are now dozens of virtualization software packages on the market with price points that are all over the map. This increase in competition gave birth to VMware Server, the next iteration of its GSX product line. What makes VMware Server so special? As you will see over the course of this book, VMware Server is a robust and versatile tool for your data center and development process. Oh, did I mention that it was free?

This chapter introduces the basic concepts of hardware virtualization, defines some early terminology, introduces you to VMware Server, and gives you a look at what life was like before virtualization.

What Is Virtualization?

Virtualization itself is a rather broad term that most commonly refers to some sort of abstraction of resources. Throughout this book I will be focusing on one specific aspect of virtualization, software virtualization.

Software Virtualization

Software virtualization is a process in which the actual physical hardware of a machine is decoupled, or abstracted if you will, from the underlying operating system by means of software. This hardware abstraction allows multiple "virtual" environments to function side by side on top of a single physical machine. Through this abstraction process each virtual machine is provided with its own hardware (RAM, CPU, Network Card, and so on) that is independent and isolated from the other virtual machines running along side it.

Figure 1-1 shows how a normal x86 machine presents its hardware to the operating system for your use, and Figure 1-2 provides a visual representation of how software virtualization takes form and how the physical machine hardware is partitioned out into multiple virtual segments for your use.

All the magic happens in the virtualization layer seen in Figure 1-2. This layer decouples your hardware and handles all allocation and utilization of the physical hardware by your virtualized machines.

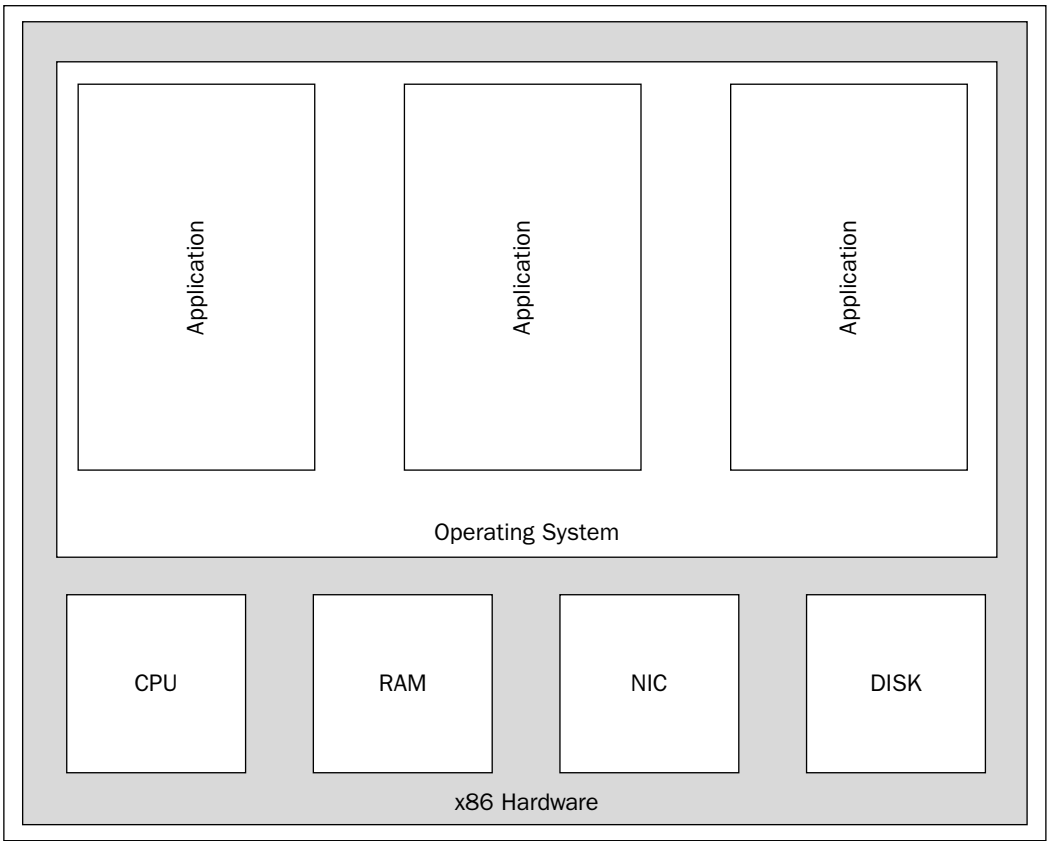

Figure 1-1

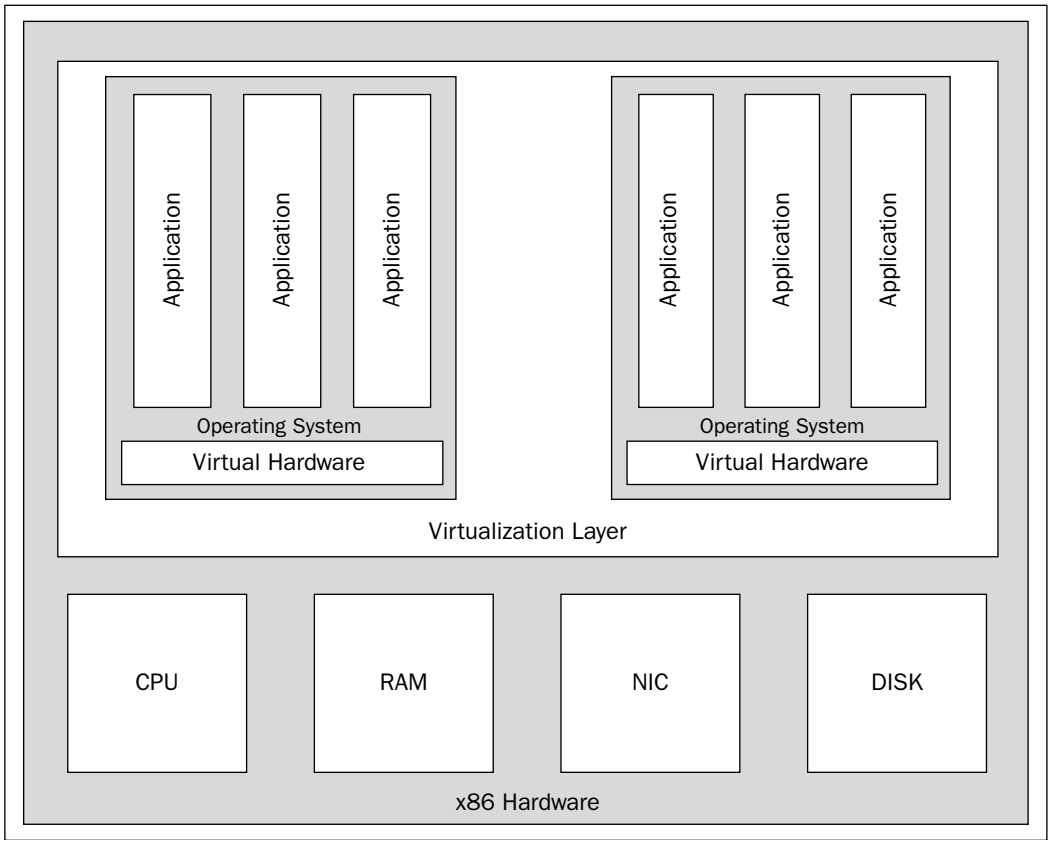

Figure 1-2

Hosted vs. Hypervisor Architecture

You can't really complete the overview of virtualization without a quick discussion of hosted versus hypervisor architecture.

- **Hosted architecture** is where your virtualization software is installed as an application onto the pre-existing host operating system. This means that your virtualization layer relies on your host operating system for device support and physical resource management. VMware Server, VMware Workstation, and Microsoft Virtual PC are good examples of a hosted architecture.
- **Hypervisor architecture** is more of a bare metal approach where there is no pre-existing operating system. The virtualization software is installed on a clean system, and it provides kernel and driver support for the raw physical hardware. VMware ESX server is a good example of virtualization utilizing hypervisor style architecture.

Figure 1-3 and Figure 1-4 provide a visual representation of the architecture types and their relation to the physical hardware.

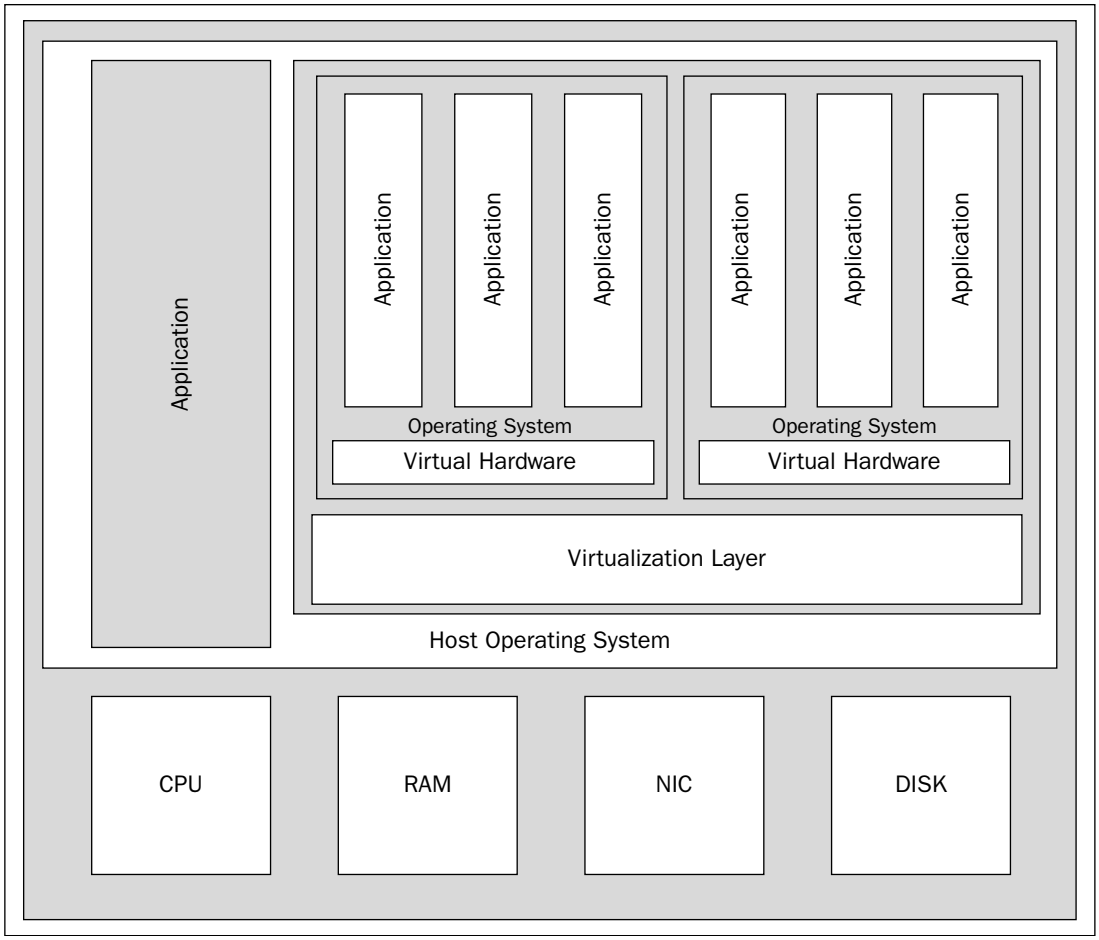

Figure 1-3

Hosted style architecture relies on the underlying operating system for hardware support. This means that theoretically as long as the underlying operating system supports the hardware it should be available to the virtualized machines. This allows for a greater variety of supported hardware. This comes at a price however, the added overhead of the physical machine's operating system. This is where the hypervisor architecture comes in. It doesn't require the physical machine to have a pre-existing operating system to be installed. In practice the virtualization software will install its own kernel and drivers for hardware. This allows for a dramatic decrease in overhead over the hosted architecture but it also seriously reduces the choices in compatible hardware. The reason is that companies, like VMware, have to choose a subset of hardware to write compatible drivers for, whereas drivers are readily available for hardware under Windows XP or Windows Server 2003 because the hardware manufacturers produce the drivers.

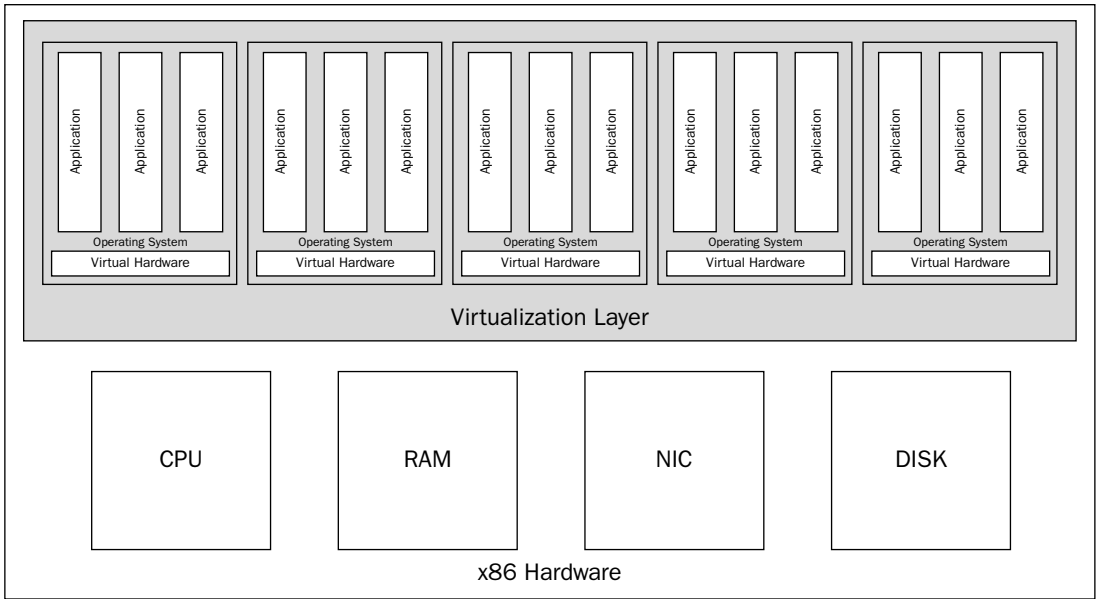

Figure 1-4

In conclusion, the following tables outline the advantages and disadvantages of each type of architecture.

Hosted Architecture	
Advantages	**Disadvantages**
Ease of deployment	Increased cost due to licensing of host software
Greater hardware support	Physical resources such as memory are shared between the host and virtual machines
Decrease cost for virtualization software	Less server utilization possible due to the additional overhead of the underlying host operating system

Hypervisor Architecture	
Advantages	**Disadvantages**
Greater resources for the virtual machines	Decreased hardware support
Decreased cost for licensing because no host operating system is required	Increased cost for virtualization software. Hypervisor software like VMware ESX server is considerably more expensive.
Increased utilization of server hardware	

Terminology, Host vs. Guest

If you remember the definition of hosted architecture you'll undoubtedly see where the term host comes from. The *host machine* is the physical machine itself, the machine where you will be installing VMware Server. The *host operating system* is the actual operating system installed on the host machine. Many times you will simply see them both referred to as just the host.

In the discussion of virtualization earlier in the chapter you learned that there are segments, or partitions, created in order to isolate and house your virtual machines. Once a new virtual machine is created, it is essentially a clean slate. Just like with a new PC you must install an operating system onto the newly created virtual machine. The *guest* refers to the operating system installed on the virtual machine. As with host, many times you'll see guest refer to both the virtual machine and its operating system.

I cover how to create virtual machines in Chapter 4.

I could spend the next several pages going over some of the other terminology used in virtualization and VMware Server; however, I will leave that for the chapters ahead. Nevertheless, I did want to stress the importance of understanding this difference between the host and guest. Though it is not a difficult concept, it is a very common one over the next several chapters.

So What Is VMware Server?

VMware Server is a multi-platform (Windows Server and Linux) application that provides host-based software virtualization. This gives you the ability to create and operate multiple x86-based virtual machines simultaneously on the same physical machine. Of course all of that is great, but what does it really do for you? Here are just a few of the advantages to running your server or development environment within virtual machines:

- Rapid deployment/decreased setup and tear down of new machines
- Multiple platforms available on a single physical machine
- Ability to move virtual machines from one server to another with no loss of information
- Increased utilization of your hardware/decreased costs

The next few sections go over each of these advantages in more detail.

Rapid Deployment

Especially in a development environment you'll find the ability to set up and tear down a set of virtual machines quickly is worth its weight in gold. This rapid setup and tear down can be accomplished by creating a library of base images. These images will contain a static install of, for example, Windows XP. Having a library of base images will allow you to quickly unpack and deploy virtual machines in a matter of minutes.

Chapter 4 covers creating and organizing a base image library.

As an example, say you have an obscure application that works right only on Windows 98. A customer calls with a serious problem to troubleshoot. In the past you would have to either dual boot a workstation into Windows 98 or have an old junk PC with Windows 98 on it for testing. Of course they aren't just lying around when you need them to be; you have to go find what door they are propping open, and what should be a 15-minute support call turns into a whole afternoon.

Now, say you receive the same phone call but this time you have a handy virtual machine already set up with Windows 98 installed. You simply connect to the VMware Server console and fire up your Windows 98 machine. You have a testing environment set up and ready within minutes, usually with the customer still on the phone. Once you finish the call you simply shut it down and forget it.

Multiple Platforms Available on a Single Physical Machine

If you're like me you spend most of your day on a Windows PC doing Windows development. The problem is that I like to tinker with new technology. I always want to try out that new Linux distro, but I never have a spare machine just lying around. This is where one of my favorite benefits of virtualization comes into the picture. If you remember earlier I talked about how the virtualization layer abstracts out the hardware from its implementation. This allows you to not just run Windows-based virtual machines, but you can run Linux, BSD, and so on. Almost anything that will run on an x86, you can run side by side with anything else in VMware Server, regardless of the host operating system. I just love that part. I would have to say the most common use I've found for this, from a developer's view, is setting up an Apache environment for web development. PHP and Perl just work so much better on an Apache installation in Linux.

The Ability to Move Virtual Machines from One Server to Another

As you will learn in Chapter 3, virtual machines in VMware Server are really nothing more than files on your hard disk. This coupled with the independent nature of the image with respect to the host hardware makes it extremely portable. In other words, there is nothing that keeps you from moving that Windows XP base image from your development VMware Server over to your VMware Workstation installation or over to a VMware ESX Server.

> *There are some minor incompatibilities between VMware Server, ESX, and Workstation images. The product you use to create the image will prompt you if a choice is made that will render the image incompatible with other versions.*

As an example, suppose you have two highly critical servers running in virtual machines on one server. Now, and we've all had this happen, a fan goes out on that server. You can't buy the fan anywhere but direct, it always seems to be that way. You can rig the machine to come up but you don't have room in your data center for a machine cracked open with a box fan propped above it. All you have to do is get that server up long enough to copy those virtual machines over to another VMware Server and switch them on. Problem solved and your users are never the wiser. Suppose this server was simply a firewall or instant messaging server. Backup images of machines like that can be kept on another VMware Server machine for easy, quick fall over.

The flexibility all the VMware products offer in this regard is outstanding. If care is taken to maintain compatibility when the virtual machines are created, you can move, mix, and match your virtual machines in any way you see fit. You can even use virtual machine images from other virtualization products like Microsoft Virtual PC and Microsoft Virtual Server. VMware Server can read and convert the images into a format compatible for use on your VMware products.

Increased Utilization of Your Hardware

Figuring the Return on Investment gives me a headache; however, it doesn't take too much to see that a $3000 server that averages a CPU load of 25 percent has a lower return than a $3000 server with an average load of 75 percent. The simple fact is your servers are probably underutilized. Take any server currently in your datacenter and monitor it for several hours. It may shock you to learn that an average server is rarely utilized over 10 to 20 percent. That leaves a bit to be desired in terms of your investment. VMware Server can keep that server doing what it was bought for, pushing bits. By consolidating your servers into a single host machine capable of multiple virtual machines, you increase the return on your investment. I haven't even figured in the decrease in power consumption, air conditioning, hardware failure, and other issues that plague a datacenter made up of several individual servers where each performs just a handful of tasks.

A perfect example that I've seen and heard from many people is that they have a piece of software that cannot be installed alongside a different piece of software; however, both are needed in the complete system. The product documentation probably tells you that you need two servers. Why? You just need virtualization. Create two virtual machines. Install the operating system and software needed on each and away you go. You save yourself the cost of an additional server and most likely still do not max out the hardware on your existing server.

Life without VMware Server

Looking at this in terms of your development process, what is life like without VMware Server? I may not be able to answer that question from your perspective, but it is certainly my goal to show you what your development process can be like with VMware Server as a part of it.

In terms of physical machines the common developer has three requirements, depending on the project of course:

- **Developer machine** — Usually the machine on your desk.
- **Servers** — Where you develop web-based code (Web Services, ASP.NET, and so on).
- **Test bed** — The bank of machines you use to test your code.

In a world where virtualization did not exist, the preceding requirements could result in a large number of physical machines. The test bed alone, if you were doing any serious stress or regression testing, could be comprised of several machines itself. Of course there can be many other types of machines added to the preceding list, but for the purposes here it covers the basic requirements.

Now, as you probably already know, I'm going to tell you the machines required for the preceding scenario would fit perfectly into virtual machines.

First, having your developers write code from within a development virtual machine has numerous advantages. In Chapter 4 I cover the advantages and disadvantages of placing your developers solely within a virtual machine. One of the biggest advantages, for the sake of discussion here, is a static environment. You can create a static development image complete with all the tools, IDE, and so on needed to function within your environment and then distribute that out to your development team. Now all of your developers are writing code on the same platform with the exact same settings and install. No more worrying about your developers installing the latest game or spyware on their work PC. Everything is static and isolated within your development image.

> *There are numerous advantages and disadvantages to developing within a virtual machine as well as a few instances where this approach might not produce the results you desire. Further information is provided in Chapter 4.*

Development servers are a perfect candidate for virtualization. Typically your development servers are not production servers, and most companies don't have the budget or extra hardware lying around to house a server just for testing code. There's almost no better use for a virtual machine.

The final common requirement is the test bed. There were two ways to handle multiplatform testing in the past. Either you had a room full of physical computers with various operating systems installed on them or you had one machine that you just spent hours in front of. More often than not in the latter situation testing would take a back seat due to the huge time requirement of testing on one machine. This is prime time stuff for VMware Server. Not only can you accomplish multiplatform testing for your code with multiple virtual machines, but I show you later in this book how you can automate the process so you can push builds out, automatically start virtual machines, and even retrieve status back from your installers and applications while testing. Imagine your nightly build starting up the necessary virtual machines, copying over the bits, and e-mailing your QA department that it's ready to be tested. That's some pretty cool stuff.

Summary

Over the past few sections you've learned about the following:

- ❑ The basics of virtualization, the types of architecture utilized, and terminology
- ❑ What VMware Server is and the advantages of using virtual machines
- ❑ The difficulties of life without VMware Server, or virtualization in general

Chapter 2 goes over the installation steps required to get VMware server up and running on your Windows or Linux-based server. In the following chapters you will take that installation and go through some ways in which you can integrate virtual machines into your existing development processes and hopefully increase your productivity.

2

Installing Your VMware Server

The installation chapter — few books are complete without it. This chapter covers installation of VMware Server for Windows and Linux host environments as well as the remote clients for each. Because both can be vastly different when it comes to software installation and configuration, each method is covered and, where appropriate, the major differences are highlighted.

My approach with this chapter is to cover each piece of the software package in order and one at a time. You learn in the coming sections that VMware Server contains four major parts:

- The server itself
- The console used to administer the server
- The web management interface
- And finally, any APIs you wish to install

For each part, the Windows and Linux installations are covered where appropriate.

> *This chapter is rather long so I encourage you to hop and skip through the sections that cover your environment and situation appropriately.*

Host Hardware Requirements

As you learned in Chapter 1 the guest virtual machines use an abstracted set of the host machine's hardware. With that being said, and like most applications these days, the faster and more robust your hardware the better your virtual machine environment will run.

VMware Server supports up to 16-way multiprocessor servers. If you take the recommended standard of no more than 4 guests per processor, this tells you that you can run a single server with up to 64 guest machines at one time. It doesn't take a whole lot to realize that is a lot of utilization for a single server. Now, most of you will never reach this magnitude of usage so for the purposes of this book, the VMware Server will be housed on a single processor server.

Processor

VMware Server requires at least a 733MHz or faster x86-based processor of one of the following types:

- **Intel** — Pentium II and higher including support for the EM64T
- **AMD** — Athlon and higher including support for the Athlon 64

For the most up-to-date processor support please consult the VMware Server Administration guide available online at the VMware web site (`www.vmware.com`*).*

Memory

Simply put, as much as you can put in. Remember that the server will have to cover not only the memory requirements for the host operating system but also the memory requirements for the guest operating systems as well. While planning your virtual infrastructure you need to factor in not only the memory requirements of the host operating system, but also the maximum memory your host OS will allow.

An example of this is if you choose Windows Server 2003 Standard as your host operating system. It requires a minimum of 128MB of RAM but allows only 4GB total system memory. Loading your server up with 64GB of RAM will do you little good in this situation. Consider this when choosing your host hardware and operating system.

VMware Server memory requirements:

- **Minimum** — 512MB
- **Maximum** —
  - 64GB for Windows and Linux hosts that have Physical Address Extension (PAE) equipped processors
  - 4GB for Windows hosts with non-PAE enabled processors
  - 2GB for Linux hosts with a 2.2.x series kernel

Again, this can't be stressed enough — plan your hardware and host operating system to allow for the maximum physical memory possible.

Hard Disk

As covered in Chapter 4, each virtual machine is made up of several files, one of which is the virtual hard disk for that machine. A good rule of thumb is whatever the operating system would take up on a normal install is what it will at the very least occupy as a virtual disk file.

A standard Windows XP Professional install takes up a little over 2GB of hard disk space out of the box.

There are a few issues to consider when choosing the hard disk space you make available on your host machine. Chapter 5 covers the different types of virtual hard disks you can create for your virtual machines, but for the sake of this topic I should cover a few basics here. When creating a new virtual machine you have two choices for the virtual hard disk, expanding or pre-allocated:

- An **expanding** virtual disk starts small and increases in size as you install your guest operating system and applications. This allows for a savings in disk space, but your performance will take a hit as a result.
- A **pre-allocated** disk starts out at a predetermined size, say 30GB, and it creates that size file right from the beginning. Regardless of what you install, the virtual disk file will always be 30GB.

Both expanding and pre-allocated require that you provide a maximum size, and they cannot grow beyond the initial value.

A third option, a physical disk or partition, is covered in Chapter 4.

Physical Address Extension (PAE)

A Physical Address Extension–enabled processor allows for a 32-bit processor to be able to access up to 64GB of system memory. The reason PAE is needed for this is due to the size of the processor address table. PAE adds support for 64-bit address lines by mapping the new 64-bit space into the available 32-bit address space. Applications still access memory based on the original 32-bit memory model while the processor ensures the memory accessed is mapped to the appropriate address. Intel provides PAE on all processors from the Pentium Pro on up with exception to some of the Pentium M mobile processor line. AMD processor support was hit and miss for a while. Now all current AMD processors support the PAE standard. Please consult the specifications for your processor to ensure you have PAE support built in before planning a system with more than 4GB total system memory.

It should also be noted that support for PAE isn't just processor based. In order for applications to access more than 4GB of memory on a 32-bit machine, the operating system has to provide its own support for the expanded memory space in tandem with the PAE-enabled processor. Microsoft Windows and most Linux distributions provide this function out of the box.

VMware Server host hard disk requirements:

- 130MB free on Windows hosts required for the installation of server, VMware Management Interface, VmPerl, VmCOM API, and VMware Server Console.

  Don't worry if you aren't familiar with the components you are installing here. VMware Management Interface, VmPerl, VmCOM, and VMware Server Console are covered in more detail in later chapters.

- 20MB free on Linux hosts required for the installation of server, VMware Management Interface, VmPerl API, and VMware Server Console.
- At least 1GB free for each guest operating system.

  Depending on the purpose of your virtual machines, the 1GB free recommendation is quite low. If you are primarily utilizing Windows-based guest machines, a recommendation of at least 10GB or more per machine is more on target. Analyze your guest environment ahead of time to determine a proper estimate of disk usage for your virtual machines. I cover in more detail my recommendations for virtual disk size and space requirements for each virtual machine in Chapter 4.

Network Controller

Because you are relying on your host operating system for hardware support, any Ethernet controller that is supported by your host operating system is supported by VMware Server. The ways in which the virtual machines can interface with the host's network controller are covered in more detail in Chapter 3.

Something to bear in mind, however, in terms of the network connection to your server is bandwidth. Because VMware Server is designed to allow for virtual machines to run on the server while being accessed remotely via a VMware Server Console, which I cover later, the network bandwidth available to your server needs to be able to support this load. I suggest you ensure your server utilizes at least a 100 Megabit or higher connection with a static IP address if possible. This will ensure you have adequate bandwidth and connectivity to your server once you move it into production.

Host Operating System Requirements

VMware Server supports both Windows- and Linux-based hosts. The differences between each are outlined in the following sections.

For the most up-to-date hardware and software requirements for VMware Server please reference the VMware Server documentation at `www.vmware.com`.

For Windows the installer will not allow you to use any unsupported operating system; however, Linux has no such restriction. You can still install the Linux version of VMware Server on any Linux-based distribution even if it is not on the support list. A word of caution, however — if you deviate from the supported Linux distribution list you may receive erratic results from the installer scripts and less than optimal performance.

Windows

First, I'll start by listing the supported host operating systems and service pack levels for Windows:

- 64-bit host computers can run the following host operating systems for Windows. Only the 64-bit versions of the operating systems will allow for 64-bit guest virtual machines:
  - Microsoft Windows Server 2003 R2 Standard, Enterprise, and Web Edition
  - Microsoft Windows Server 2003 SP1 Standard, Enterprise, and Web Edition
- 32-bit host computers can run the following host operating systems for Windows:
  - Microsoft Windows Server 2003 R2 Standard, Enterprise, and Web Edition
  - Microsoft Windows Server 2003 Standard, Enterprise, and Web Edition including SP1
  - Microsoft Windows Server 2003 Small Business Editions including SP1
  - Microsoft Windows 2000 Server, SP3 and SP4
  - Microsoft Windows 2000 Advanced Server, SP3 and SP4

The base installations of the preceding Windows Server operating systems already contain the prerequisites necessary to run VMware Server, with the exception that VMware Management Interface requires the installation of Internet Information Services (IIS) 5.0 or 6.0.

> *Microsoft Windows XP is not supported as a host operating system in any way. For the most up-to-date listing please consult the VMware Server Administration guide available online at the VMware web site (*`www.vmware.com`*).*
>
> *VMware Management Interface is an optional item and is not required for basic VMware Server operation. It does, however, offer some advantages that make it a valuable tool.*

Linux

Because of the vast array of Linux-flavored operating systems offered, the following list is rather lengthy, and due to the rapid release of Linux kernels, official support from VMware tends to lag behind. You can find an updated list of supported operating systems and kernels on the VMware web site at www.vmware.com.

> *Due to the rapid change in Linux distributions the list that follows will quickly become out of date. Some of the operating systems and kernel versions listed are only supported in a limited experimental fashion. As always, please consult the VMware Server Administration guide for the most up-to-date list of supported operating systems and kernels.*

- 64-bit host computers can run the following host operating systems for Linux. Only the 64-bit versions of the operating systems will allow for 64-bit guest virtual machines:
  - Red Hat Enterprise Linux 4.0 AS, ES, and WS including U3
  - Red Hat Enterprise Linux 3.0 AS, ES, and WS, stock kernel 2.4.21, update 2.4.21-15, and updates 6 and 7. Update 8 is experimental support only.

- SuSE Linux Enterprise Server 10 (experimental support only)
- SuSE Linux Enterprise Server 9 RTM, SP1, SP2, and SP3
- SuSE Linux 10.x
- SuSE Linux 9.3
- SuSE Linux 9.2 RTM and SP1
- SuSE Linux 9.1 with the stock 2.6.4-52 kernel
- Mandriva Linux 2006
- Ubuntu Linux 5.04 and 5.10
- Ubuntu Linux 6.06 (experimental support)

- 32-bit host computers can run the following host operating systems for Linux:
  - Red Hat Enterprise Linux 4.0 RTM, updates 1, 2, and 3
  - Red Hat Enterprise Linux 3.0 RTM, updates 1 thru 7
  - Red Hat Enterprise Linux 3.0 Update 8 (experimental support)
  - Red Hat Enterprise Linux 2.1 stock 2.4.9-e3 kernel
  - Red Hat Linux 9.0 stock 2.4.20-8 and upgrade 2.4.20-20.9 kernels
  - Red Hat Linux 8.0 stock 2.4.9-31 kernel
  - Red Hat Linux 7.3 stock 2.4.18 kernel
  - Red Hat Linux 7.2 stock 2.4.7-10 and upgrades 2.4.9-7, 2.4.9-13, 2.4.9-21, and 2.4.9-31 kernels
  - Red Hat Linux 7.1 stock 2.4.2-2 and upgrade 2.4.3-12 kernels
  - Red Hat Linux 7.0 stock 2.2.16-22 and upgrade 2.2.17-14 kernels
  - SuSE Linux Enterprise Server 10 (experimental support)
  - SuSE Linux Enterprise Server 9 RTM, SP1, SP2, and SP3
  - SuSE Linux Enterprise Server 8 stock 2.4.19
  - SuSE Linux 10
  - SuSE Linux 9.3
  - SuSE Linux 9.2
  - SuSE Linux 9.1 stock 2.6.4-52
  - SuSE Linux 9.0 stock 2.4.21-99
  - SuSE Linux 8.2 stock 2.4.20
  - SuSE Linux 7.3
  - Mandriva Linux 2006
  - Mandrake Linux 10.1

- Mandrake Linux 9.0 stock 2.4.19
- Ubuntu Linux 5.04 and 5.10
- Ubuntu 6.06

The list is long and rather picky when it comes to supported kernels. It should be noted that this is not a list of what VMware Server will work on as much as a list of what VMware supports. If support from VMware is of absolute importance to you, sticking to this list is a must.

Unlike Windows the base installs of the previously mentioned Linux distributions may not necessarily be enough, depending of course on the distribution in question. The following is a list of some absolute requirements across all of the Linux distributions:

- Kernel 2.2.14-5.0 is not supported on any distribution due to problems with memory corruption under heavy load.
- `glibc` version 2.1 or higher is required with the `libXpm.so` library.
- `inetd` is required for the VMware Server Console and VMware Management Interface.
- The SCSI Generic (`sg.o`) driver version 2.1.36 is required to use generic SCSI devices in your virtual machines.

More on SCSI and other hardware devices for your virtual machines is covered in Chapter 3.

- Perl 5.005x or higher is required for the VmPerl API.
- An X Server is required to run the VMware Server Console.

Installation of the preceding requirements is beyond the scope of this book. Consult your distribution's documentation for more information.

Remote Client Requirements

The *remote client* is how VMware refers to the use of the VMware Server Console or VMware APIs on remote Windows- or Linux-based machines. The VMware Server Console is an application used to interface with the VMware Server. The console is where you create new virtual machines, power on and off existing virtual machines, and interface with the guest operating systems. Whereas the server console usually exists on the host machine itself, the console can be installed on several different remote platforms. This console is how the users of your virtual machines will connect to the virtual machines from their workstations. You'll start to see this come together more in later chapters. For now I want to cover the supported platforms for the server console.

Hardware

- x86-based computer
- 266 MHz or faster processor
- 64MB of RAM minimum

- 30MB for Windows installations or 60MB for Linux installations
- 17MB for the installation of the VmPerl and VmCOM APIs on Windows and 14MB for the installation of the VmPerl API on Linux

Operating System

Windows

- Windows Server 2003 x64, Standard, Enterprise, and Web Edition
- Windows XP Professional and XP Home Edition with SP1 and SP2
- Windows 2000 Professional, Server, and Advanced Server with SP1 thru SP4
- Windows NT 4.0 Workstation and Server with SP6a and Internet Explorer 6.0 installed

Linux

- Any Linux installation with `glibc` version 2.1 or higher and one of the following kernels:
  - Any kernel 2.0.32 or higher in the 2.0.x series or a kernel in the 2.2.x, 2.4.x, or 2.6.x series for all single processor systems
  - Any kernel in the 2.2.x, 2.4.x, or 2.6.x series for multiprocessor systems

As I mentioned before, kernel 2.2.14-5.0 is not supported.

- Perl 5.005x or higher is required for the VmPerl API.
- An X Server is required to run the VMware Server Console.

You'll notice the hardware and software requirements for the remote client are rather light as compared to the requirements for the server. All your remote client is responsible for is connecting to and displaying your virtual machines. The workload of the virtual machine is all housed on your server. This is what makes VMware Server so useful. If the value of this is not already obvious it will become more apparent as you progress throughout the book.

Installing VMware Server

Because VMware Server has two major forks in host operating system compatibility, I cover each procedure in its own section where appropriate.

You cannot install VMware Server on a computer with another VMware product. You will need to uninstall the previous product prior to installing VMware Server. For Windows uninstalls use Add and Remove programs. For Linux, usually the VMware uninstall scripts are located in `/usr/bin` and called `vmware-uninstall.pl`.

Installing VMware Server on Windows

All Windows-based procedures outlined in this book utilize Windows Server 2003 Standard with SP1 and IIS 6.0. The installation steps that follow differ only slightly, if any, from the installation procedures for the other supported Windows Server–based platforms.

The Windows installation provides probably an easier install experience compared to the Linux-based install. The following section outlines the steps needed to install VMware Server, the VMware Server Console, VmPerl, VmCOM APIs, and the VMware Management Interface.

The following assumes you have already installed Windows Server 2003 with IIS 6.0. If you need assistance with the installation and configuration of Windows Server 2003 or IIS 6.0, please consult the Microsoft documentation.

1. First you need to obtain the VMware Server binaries from the VMware web site. You can download these at `www.vmware.com/download/server`. The binary you are interested in is the full master install for VMware Server. The client packages that are also available are not necessary at this time. They are addressed in later sections.

You will need to register with VMware prior to the download in order to obtain the serial number required later in the install process.

2. After downloading the installation package, double-click it to begin the install process. You will need administrative privileges on the host machine in order to complete the install. You should see a screen similar to the one in Figure 2-1. Click Next.

VMware Server
Welcome to the installation wizard for VMware Server
The installation wizard will install VMware Server components on your computer. To continue, click Next.
VMware Server
WARNING: This program is protected by copyright law and international treaties.
< Back
Next >
Cancel

Figure 2-1

3. The next screen contains the software's licensing terms, shown in Figure 2-2. Read and, if you so choose, accept the software licensing terms and click Next.

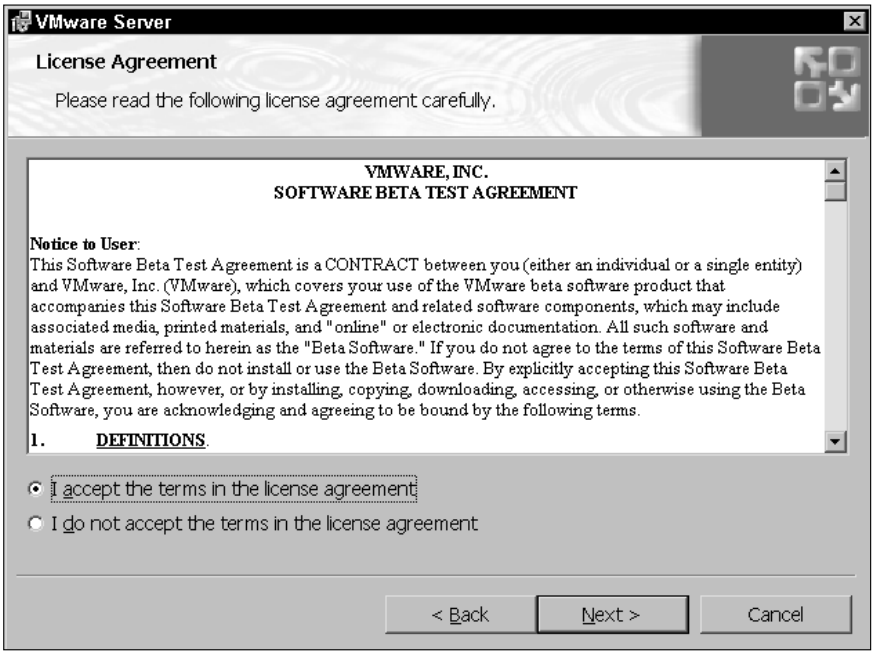
VMware Server
License Agreement
Please read the following license agreement carefully.
VMWARE, INC.
SOFTWARE BETA TEST AGREEMENT
Notice to User:
This Software Beta Test Agreement is a CONTRACT between you (either an individual or a single entity) and VMware, Inc. (VMware), which covers your use of the VMware beta software product that accompanies this Software Beta Test Agreement and related software components, which may include associated media, printed materials, and "online" or electronic documentation. All such software and materials are referred to herein as the "Beta Software." If you do not agree to the terms of this Software Beta Test Agreement, then do not install or use the Beta Software. By explicitly accepting this Software Beta Test Agreement, however, or by installing, copying, downloading, accessing, or otherwise using the Beta Software, you are acknowledging and agreeing to be bound by the following terms.
1. DEFINITIONS.
I accept the terms in the license agreement
I do not accept the terms in the license agreement
< Back
Next >
Cancel

Figure 2-2

4. Next you want to select the Setup Type shown in Figure 2-3. I'm going to select the Custom Setup Type so you can see what the installer is doing. Select Custom and click Next.

5. As you can see in Figure 2-4, you are installing the server, its consoles, and the APIs. If you do not have IIS installed or are not interested in installing the web management interface, deselect it now by clicking the drive icon next to VMware Management Interface and selecting "Do not install."

6. You may, depending on your operating system and any changes you may have made to it prior to starting the install, receive a message similar to the one in Figure 2-5. It is important, but not required, to disable CD-ROM autorun as suggested. If you choose not to disable autorun you may experience some strange behavior in your virtual machine if you insert a CD-ROM into your host server. This behavior may result in both the host server and the virtual machine attempting to autorun the CD-ROM at the same time. The result is more annoying than harmful.

   If the message in Figure 2-5 appears, make sure the checkbox is selected and click Next. If the preceding screen did not appear or if you already selected Next to move past it you should see the ready to install screen. Select Install to begin the installation.

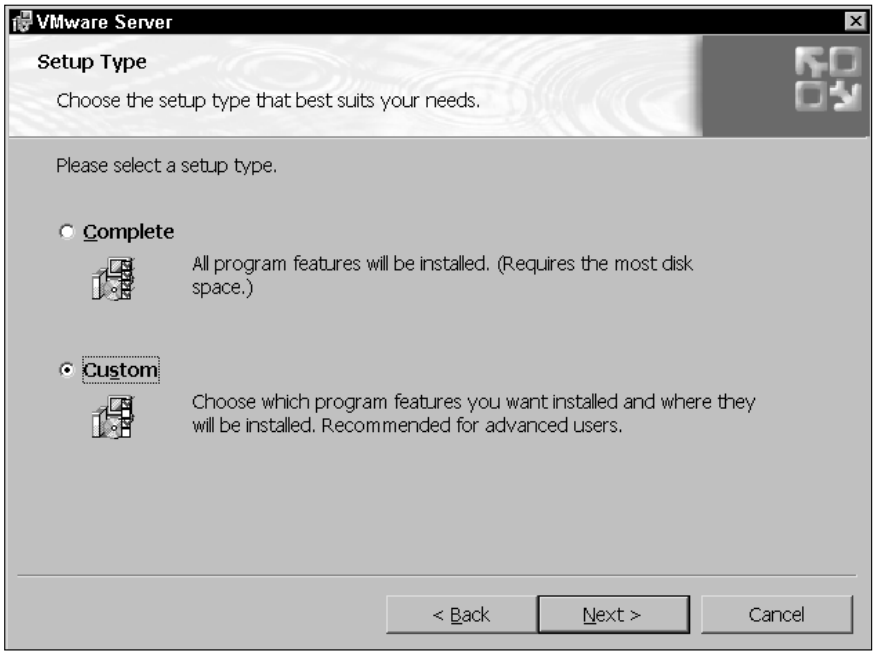
VMware Server
Setup Type
Choose the setup type that best suits your needs.
Please select a setup type.
Complete
All program features will be installed. (Requires the most disk space.)
Custom
Choose which program features you want installed and where they will be installed. Recommended for advanced users.
< Back
Next >
Cancel

Figure 2-3

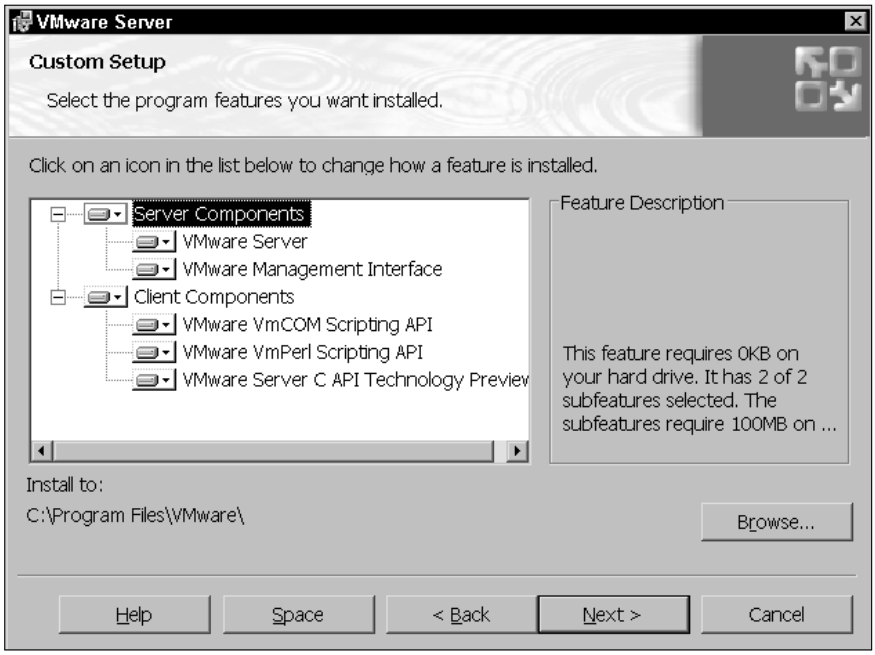
VMware Server
Custom Setup
Select the program features you want installed.
Click on an icon in the list below to change how a feature is installed.
Server Components
VMware Server
VMware Management Interface
Client Components
VMware VmCOM Scripting API
VMware VmPerl Scripting API
VMware Server C API Technology Preview
Feature Description
This feature requires 0KB on your hard drive. It has 2 of 2 subfeatures selected. The subfeatures require 100MB on ...
Install to:
C:\Program Files\VMware\
Browse...
Help
Space
< Back
Next >
Cancel

Figure 2-4

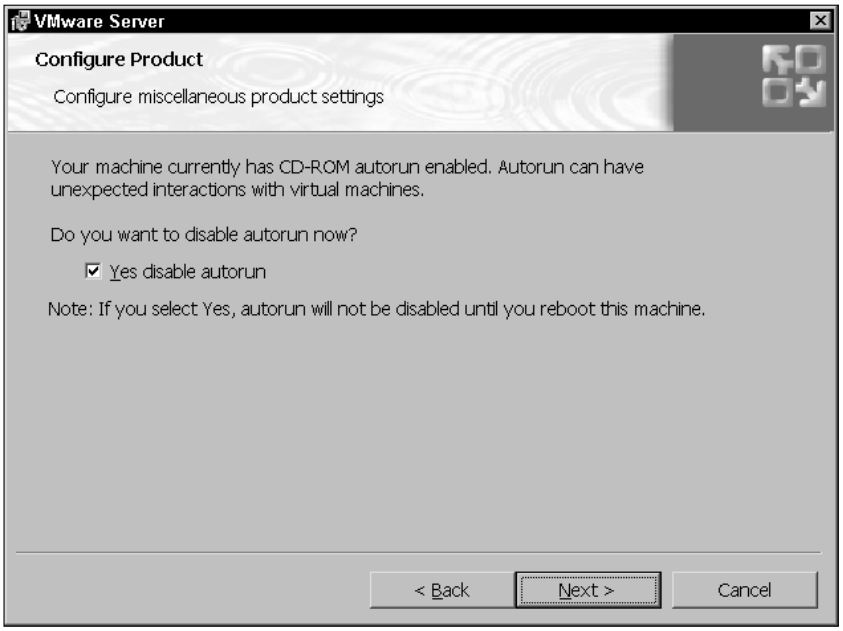
VMware Server
Configure Product
Configure miscellaneous product settings
Your machine currently has CD-ROM autorun enabled. Autorun can have unexpected interactions with virtual machines.
Do you want to disable autorun now?
Yes disable autorun
Note: If you select Yes, autorun will not be disabled until you reboot this machine.
< Back
Next >
Cancel

Figure 2-5

7. Once the installation files are copied you will be prompted for the serial number that was provided to you during the registration process with VMware. You will be presented with a screen similar to the one shown in Figure 2-6. You can skip entering the serial number at this time; however, you will be prompted for it prior to being able to power on any of your virtual machines from the server console.

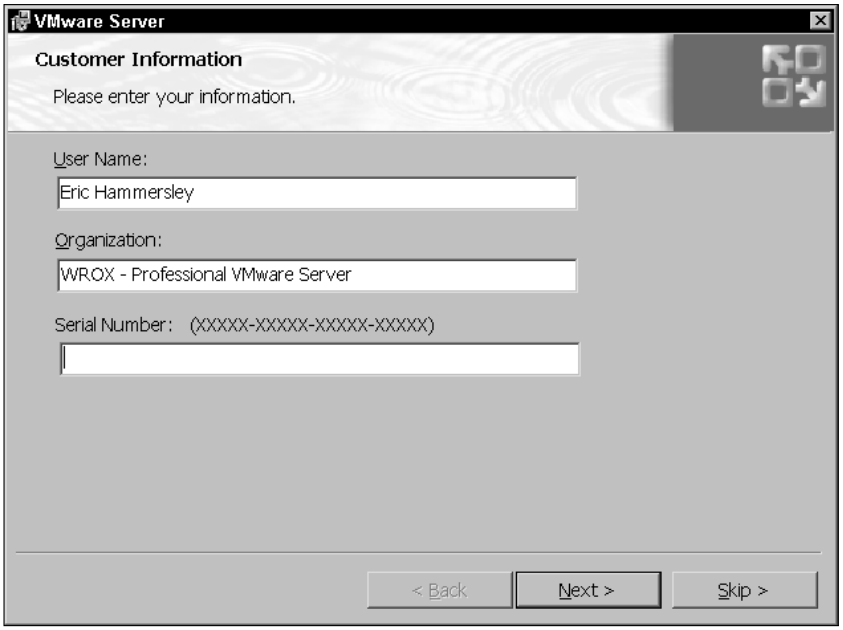

Figure 2-6

The installation of VMware Server is now complete. You should now have a new folder in your Start menu like the one pictured in Figure 2-7 as well as a new desktop icon. You can fire it up and see if everything is working.

Once the VMware Server Console loads you will be prompted to select what server to connect to. The default selection is localhost. Select localhost and click OK to reveal Figure 2-8, the main VMware Server Console.

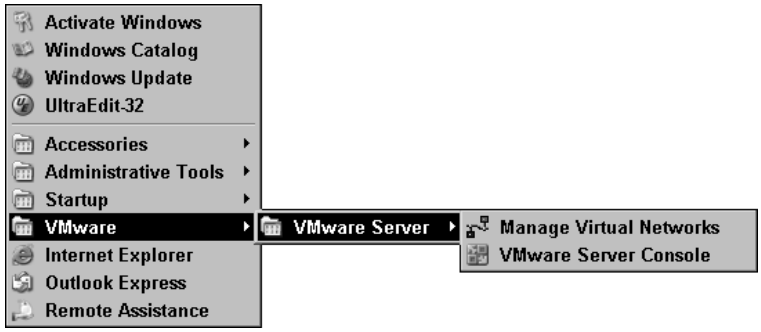

Figure 2-7

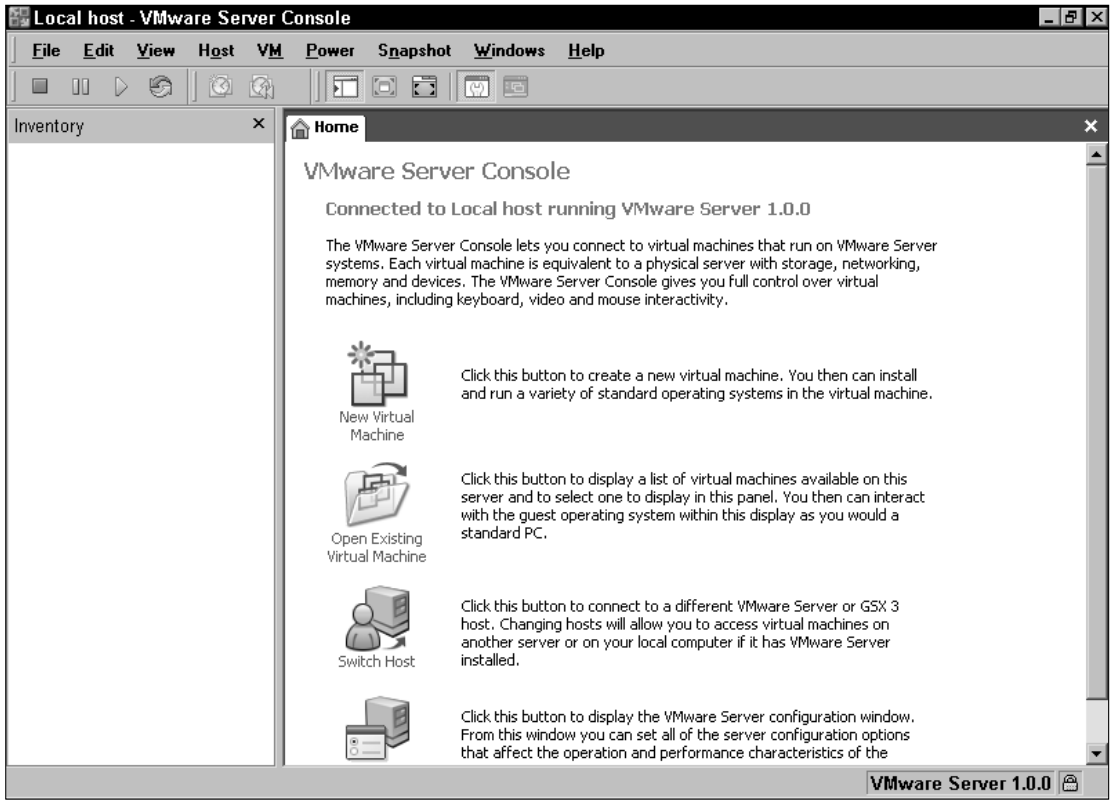
Local host - VMware Server Console
File Edit View Host VM Power Snapshot Windows Help
Inventory
Home
VMware Server Console
Connected to Local host running VMware Server 1.0.0
The VMware Server Console lets you connect to virtual machines that run on VMware Server systems. Each virtual machine is equivalent to a physical server with storage, networking, memory and devices. The VMware Server Console gives you full control over virtual machines, including keyboard, video and mouse interactivity.
New Virtual Machine
Click this button to create a new virtual machine. You then can install and run a variety of standard operating systems in the virtual machine.
Open Existing Virtual Machine
Click this button to display a list of virtual machines available on this server and to select one to display in this panel. You then can interact with the guest operating system within this display as you would a standard PC.
Switch Host
Click this button to connect to a different VMware Server or GSX 3 host. Changing hosts will allow you to access virtual machines on another server or on your local computer if it has VMware Server installed.
Click this button to display the VMware Server configuration window. From this window you can set all of the server configuration options that affect the operation and performance characteristics of the
VMware Server 1.0.0

Figure 2-8

The second shortcut seen in Figure 2-7, Manage Virtual Networks, is a bit more complicated. For now just be aware that it is there; Chapter 3 covers it in more detail.

The VMware Server installation is now complete. The next section moves on to installing VMware Server in Linux.

Installing VMware Server on Linux

Software installation on Linux has come a long way over the past several years and VMware does an excellent job of minimizing the effort by providing an almost automated install on most Linux distributions.

Downloading and Installing the VMware Server Software

First, I want to cover what I will be using in the way of the Linux host operating system and its configuration.

All Linux-based procedures outlined in this book utilize Red Hat Enterprise Linux 4.0 WS with the 2.6.9-34.EL kernel.

The procedures in this section assume you have already installed a supported Linux distribution and the required prerequisites outlined in previous sections. If you need assistance with the installation and configuration of the Linux distribution you choose, please consult the documentation for that distribution.

1. First you need to obtain the VMware Server binaries from the VMware web site at `www.vmware.com/download/server`. The item you are interested in is the VMware Server core application. You will notice there is both a tar and rpm binary available for download. This section covers an installation utilizing the tar archive because it is the method used across the most distributions and is more likely to cover everyone's installation scenario the best.

You will need to register with VMware prior to the download in order to obtain the serial number required later in the install process.

2. Once you have the tar archive downloaded, open a console and extract the archive in the current directory. For now I'll assume you downloaded the archive into your home directory. Type the command that follows:

```
tar zxf VMware-server-<version>.tar.gz
```

Insert the version number you downloaded into the `<version>` *tag in the preceding code line.*

After a few moments the command will extract the contents of the tar into a folder named `vmware-server-distrib` in the same location as your archive. Because you are working in your home directory, it will be in your `~/vmware-server-distrib` folder.

The ~ directory is a shortcut that refers to your home directory. The actual path may vary by distribution but the shortcut itself is somewhat universal. On Red Hat it translates to the `/home/<username>` *directory.*

3. The installer requires root access in order to compile and add modules into your running kernel among other things. Type the following command to switch to the root. You will need to know the root password for your system:

```
su -
```

Tar and GZip Archives

Tar, or *tape archive*, stems from its original use, which was to archive files for storage on magnetic tape. In Linux the use of tar is widespread and allows for convenient archival of file and directory structures, much like zip files on Windows only without the compression.

A tar archive with the `.gz` extension has been compressed utilizing GZip. GZip (GNU zip) is a compression utility installed on most Linux distributions. It functions in many ways like zip compression tools do on Windows, if you are more familiar with them. It simply compresses the tar archive, or any file for that matter, into a more compact form.

The switches used in the preceding code line, `zxf`, tell the `tar` command to un-gzip the archive (`z`), expand the archive (`x`), and tell it that you are specifying the file name (`f`).

If you wanted to do just the opposite and create a tar archive and compress it with GZip, you would issue the following command in the console:

```
tar czf <files to compact> <file name of resulting archive>
```

4. After you issue the `su -` command, your home directory, where you downloaded and extracted your files for installation, will no longer be the current directory. You will now find yourself in the root user's home directory. To switch back to your user directory where you left your files, issue the command that follows, inserting your user name in place of the `<username>` tag:

```
cd /home/<username>/vmware-server-distrib
```

Note that you cannot use the ~ modifier in this case because it now refers to the root user's home directory. If your distribution stores users' home directories somewhere other than `/home`*, insert it in lieu of the one provided.*

5. It's time to kick off the install script and get things running. Located inside the extracted folder you'll find a file called `vmware-install.pl`. This Perl script will automate most of the install for you. Start the install script with the following command in the console:

```
./vmware-install.pl
```

The use of the `./` *that precedes this command is required. If you receive a* `command not found` *error, verify you are in the correct directory and that you prefixed the command properly.*

6. Once the install starts you will be asked a series of questions regarding the location to install certain files. The VMware installer does a pretty good job of figuring out where everything needs to be placed based on your Linux distribution. You should see something similar to the following text:

```
Creating a new installer database using the tar3 format.

Installing the content of the package.

In which directory do you want to install the binary files?
[/usr/bin]
```

Throughout the install sequence default answers are provided inside square brackets like the ones shown in the preceding code block. You can simply press Enter if the default is acceptable or type a new path and press Enter. Changing the default path provided requires a thorough understanding of your system and should be done only in special circumstances.

The installer will continue to ask you where to place various files, libraries, and documentation manuals. The default paths provided should be used and considered accurate for your distribution. Again, unless you have a thorough understanding of your distribution or you are installing under some special circumstances, you should accept the defaults provided.

```
What is the directory that contains the init directories (rc0.d to rc6.d)?
[/etc/rc.d]

What is the directory that contains the init scripts?
[/etc/rc.d/init.d]

In which directory do you want to install the daemon files?
[/usr/sbin]

In which directory do you want to install the library files?
[/usr/lib/vmware]
```

7. After you select the default location to install the library files, the installer will ask you if it should create the necessary directories that do not already exist. Unless you have installed a previous version of VMware on your host machine, this is normal. Allow the creation and continue on.

```
The path "/usr/lib/vmware" does not exist currently.  This program is going to
create it, including needed parent directories.  Is this what you want?
[yes]

In which directory do you want to install the manual files?
[/usr/share/man]

In which directory do you want to install the documentation files?
[/usr/share/doc/vmware]
```

8. Once again the install will prompt you for permission to create the directories needed to install the documentation files. Like before, allow the installer to create the necessary directories and continue.

```
The path "/usr/share/doc/vmware" does not exist currently.  This program is going
to create it, including needed parent directories.  Is this what you want?
[yes]
```

The main install is now complete and you should see something similar to what follows:

```
The installation of VMware Server <build number> for Linux competed successfully.
You can decide to remove this software from your system at any time by invoking the
following command: "/usr/bin/vmware-uninstall.pl".
```

The su (switch user) Command

Unlike Windows, most Linux distributions are configured for the average user to have a non-privileged account, or in Windows terms non-administrator. For ordinary users on Linux to be able to perform root-level operations they would either have to log in to another terminal as root or, more conveniently, use the switch user command.

The `su -` command along with the root password will allow you to temporarily log in during that console session as root. The `-` argument tells the command to load the root user's environment. This includes file paths and other environment variables necessary to perform certain tasks.

If you find commands keep returning `command not found` make sure you used the `-` argument with `su`.

When you are finished with root, be sure to type `exit` before leaving the console unattended.

Performing Some Initial Configuration

Now that the VMware Server is installed, a few items need configuration before things start to turn. Server configuration is handled through a script called `vmware-config.pl` located in the `/usr/bin` directory. The installer should be offering to run that configuration script for you, so who are you to argue?

```
Before running VMware Server for the first time, you need to configure it by
invoking the following command: "/usr/bin/vmware-config.pl".  Do you want this
program to invoke the command for you now? [yes]
```

1. Follow the prompts to read and accept the End User License Agreement and continue. You can use the space bar to advance through the license agreement by page or the Enter key to advance a single line at a time. Once you reach the end you will be prompted to type either **yes** or **no** to continue on.

2. Now the installer will copy over some of the GUI components for the server console. Like before, accept the default location provided.

```
In which directory do you want to install the mime type icons?
[/usr/share/icons]

What directory contains your desktop menu entry files?  These files have a .desktop
file extension. [/usr/share/applications]

In which directory do you want to install the application's icon?
[/usr/share/pixmaps]
```

3. Next you get into installing modules into your kernel:

   - If you are using a distribution and kernel from the supported list mentioned previously, then most likely the installer will download a precompiled module for you. If you are, you will see a message similar to the following:

```
Trying to find a suitable vmmon module for your running kernel.

The module bld-2.6.9-5.EL-i686-RHEL4 loads perfectly in the running kernel.
```

- If you are not one of the lucky ones — and it's all right if you aren't — you will see the following message:

```
Trying to find a suitable vmmon module for your running kernel.

None of the pre-built vmmon modules for VMware Server is suitable for your running
kernel.  Do you want this program to try to build the vmmon module for your system
(you need to have a C compiler installed on your system)? [yes]
```

If you receive this message, flip forward to the section titled "Having to Compile Your Own Kernel Modules" in the troubleshooting material toward the end of the chapter. Keep your thumb here, however, because you'll be back once you complete that section.

Once the build is complete, if you had to build your own module, you should receive a message stating the modules loaded successfully:

```
The module loads perfectly in the running kernel.
```

4. Next, the install will move into the network configuration, as observed by the following text:

```
Do you want networking for your virtual machines? (yes/no/help) [yes]
```

Virtual networking can get a little complicated, especially if you aren't that familiar with networking in general. The first prompt you will see gives you three options. I cover networking with your virtual machines later in Chapter 3, or if you are starving for information now, feel free to choose the `help` option offered in the preceding code line. The answer you want is `yes` unless you want to view the help pages provided.

Without getting into a large amount of detail here you will have three basic options for networking your virtual machines: Bridged, NAT, and Host-only:

- If you enable networking you will automatically have bridged networking. **Bridged networking** is simply extending your network interface to the virtual machine, like a bridge. The virtual machine will need to be assigned its own IP address on your network or have a Dynamic Host Configuration Protocol (DHCP) server available to issue it one. It will, as far as the network is concerned, just be another machine with its own IP and MAC address. This code line should show you where you are at:

```
Configuring a bridged network for vmnet0.
```

VMware Server identifies its virtual networks by an identifier like you see in the preceding line. Each virtual network is identified as `vnnet#` where the `#` is a number from 0–9. You can create many different kinds of elaborate networking setups, but for now just stick with a bridged network on `vmnet0`.

If you have more than one network adapter on your host you may be presented with a question like the one that follows:

```
Your computer has multiple Ethernet network interfaces available: eth0, eth1.
Which one do you want to bridge to vmnet0? [eth0]
```

Understanding how the network adapters are set up on your system is important here. In this case `eth0` is the network adapter connected to your network, and `eth1` is simply a spare adapter. If you choose the wrong adapter you will experience no connectivity in your virtual machines. Never fear, however; you can come back and change your setup later. For now stick with `eth0`.

Once you select your adapter, if you have more than one, you will receive a confirmation that everything is set up for your bridged network:

```
The following bridged networks have been defined:

. vmnet0 is bridged to eth0

Do you wish to configure another bridged network? (yes/no) [no]
```

For now you are done with bridged adapters, so choose the default of `no` if you are asked to configure another bridged adapter. It is possible under some circumstances, like only one physical Ethernet adapter, that this question will not be presented.

- Now you're to NAT. **Network Address Translation (NAT)** is simply where you have only one IP address available on your host machine. An example would be your broadband connection at home. Typically the broadband provider gives you only one IP address for accessing its network. Because you learned previously that bridged networking requires that you either give your virtual machines its own static IP address on your network or make a network DHCP server available to it, bridged becomes a poor choice if you have only one IP address available, like in the broadband example. Go ahead and configure NAT networking for your virtual machines now. Choose the default of `yes` and continue on.

```
Do you want to be able to use NAT networking in your virtual machines? (yes/no)
[yes]
```

You'll notice that the installer automatically assigns your first NAT network to `vmnet8`. Now you have `vmnet0` and `vmnet8` available to your virtual machines. You'll see more about how to use them later.

```
Configuring a NAT network for vmnet8.
Do you want this program to probe for an unused private subnet? (yes/no/help) [yes]
```

Now the installer needs to probe your network for an unused private subnet to use for the NAT network. All that happens here is that the installer looks at your physical network and determines where an empty subnet exists. For example, if your network uses an address in the 192.168.0.0 subnet, the install will not choose that address space for its subnet. Results here will vary between installations, but ultimately the installer will choose a subnet within one of the published private IP ranges. For now select Yes and accept the IP subnet it provides and continue. I cover private IP addresses and how to change the subnets used for your interfaces more in Chapter 3.

```
Probing for an unused private subnet (this can take some time)...

The subnet 172.16.127.0/255.255.255.0 appears to be unused.

The following NAT networks have been defined:

. vmnet8 is a NAT network on private subnet 172.16.127.0.

Do you wish to configure another NAT network? (yes/no) [no]
```

You will most likely not receive the same private IP subnet as indicated in the preceding code block. All that is important at this point is that it finds an unused subnet to use for your NAT network.

You can configure more NAT networks for your virtual machines at this point; however, for now just stick with the one.

- **Host-only networking** consists of a separate "virtual" network adapter on the host that exists only to network with the guest, or virtual machine. This host adapter is not connected to any outside network and exists only to provide connectivity between the host and the guest. In fact, the only way to connect this host-to-guest network to an outside network is via some sort of software, like a proxy server. The fact is you probably won't do that because NAT and bridged exist for that reason. Host-only serves its purpose when you just want to connect the host and guest and nothing else. Like with NAT the installer will probe for another, different unused private subnet.

```
Do you want to be able to use host-only networking in your virtual machines?
[yes]

Configuring a host-only network for vmnet1.

Do you want this program to probe for an unused private subnet? (yes/no/help) [yes]

Probing for an unused private subnet (this can take some time)...

The subnet 172.16.183.0/255.255.255.0 appears to be unused.

The following host-only networks have been defined:

. vmnet1 is a host-only network on private subnet 172.16.183.0.

Do you wish to configure another host-only network? (yes/no) [no]
```

You have had enough with the network configuration for now. Choose the default of `no` and continue.

5. Now you find yourself back at kernel modules, only this time it's looking for the vmnet, or network, kernel module.

   - Like before, if you are using a distribution and kernel from the supported list mentioned earlier in the chapter, then most likely the installer will download a precompiled module for you. If you are, you will see a message similar to the one that follows:

```
Trying to find a suitable vmnet module for your running kernel.

The module bld-2.6.9-5.EL-i686-RHEL4 loads perfectly in the running kernel.
```

- If not, the installer will remember where the C compiler and kernel header files are from the previous build and take off building the vmnet module on its own. Once it's complete, you are looking for the following message:

```
The module loads perfectly in the running kernel.
```

6. Next the installer will start to configure remote connection for the server console. This is how your workstations on the network will communicate with VMware Server in order to create and utilize the virtual machines. The default port of this communication is 902/tcp. If port 902 is free and nothing else is running on it, you will receive the following message:

```
Please specify a port for remote console connections to use. [902]
```

If, however, something is running on that port or for some reason the install thinks the default port is not free, you will receive a message stating this and it will recommend an alternate port:

```
The default port : 902 is not free.  We have selected a suitable alternative port
for VMware Server use.  You may override this value now.  Remember to use this port
when installing remote clients on other machines.  Please specify a port for remote
console connections to use [904]
```

The port it recommends, in this example port 904, may not necessarily be the same port your installer chooses as an alternate. The simple fact is that it doesn't matter what you set this port for. You will, however, find it easier if you use the default port because the VMware Server Console GUI will default to looking at port 902. Using the nonstandard port is fine, but you'll need to remember this when accessing your server via the VMware Server Console from a remote client.

If you want to use the default port, but the install says it is not free, flip forward to "Verifying Port 902 Is Free" in the troubleshooting section at the end of this chapter. If in fact you determine that port 902 is free you can override the recommendation, type 902 and press Enter.

If you have a firewall installed on your Linux host, you will need to enable the port you choose in the preceding example to allow access to remote clients. This step is required regardless of the port used. See "Enabling the MUI and Console Ports in Your Firewall" in the troubleshooting section at the end of this chapter for more information.

7. After you finish choosing your port, the installer writes this to a startup script. Now, you'd never notice this unless you get the following error:

```
Unable to find any instance of the super-server "inetd" or "xinetd".  It is
possible that you do not have one of these packages installed on this machine.
Please install "inetd" or "xinetd".

If you do have "inetd" or "xinetd" installed, make sure that /etc/inetd.conf or
/etc/xinetd.d exists.
The configuration will continue, but you should re-run /usr/bin/vmware-config.pl
after you fix the super-server.

Hit enter to continue.
```

See the section "Installing xinetd" in the troubleshooting discussion at the end of this chapter for further information. The install script will end after you hit Enter, and you will have to start the configuration script over. The troubleshooting section will guide you through this.

If you didn't receive the preceding error, the installer will build the VMware VmPerl Scripting API for installation:

```
Building the VMware VmPerl Scripting API.

Using compiler "/usr/bin/gcc".  Use environment variable CC to override.

Installing the VMware VmPerl Scripting API.

The installation of the VMware VmPerl Scripting API succeeded.
```

8. Next, the installer will generate an SSL certificate used to secure communications between the VMware Console and the server. It will then prompt you for the location to store your virtual machines. This location will be owned by root; however when you create your virtual machines via the console interface, they will be owned by the user who created them. Chapter 3 contains more on security and multiple users accessing the virtual machines in Linux. For now accept the default provided:

```
Generating SSL Server Certificate

In which directory do you want to keep your virtual machine files?
[/var/lib/vmware/Virtual Machines]
```

Some prefer setting the final directory without a space as in "VirutalMachines" or even "virtualmachines." This really adds nothing other than ease of use from the command line. It becomes awkward having to capitalize and escape spaces on the Linux command line. Regardless of the method you choose the functionality will remain the same.

9. Like most commercial applications you need a serial number. This number should have been provided to you via e-mail when you registered at the time you downloaded the binaries from VMware. Enter the serial number here that was issued to you for the Linux install. If you want you can skip entering the serial number at this time and enter it later through the server console by choosing the Help➪Enter Serial Number menu item.

```
Please enter your 20-character serial number.


Type XXXXX-XXXXX-XXXXX-XXXXX or 'Enter' to cancel:
```

Be sure to enter your serial number exactly as it was sent to you. It is case-sensitive and requires you to insert the hyphens where appropriate.

If everything went well you should be greeted with a friendly message stating that VMware Server for Linux is running and completed successfully. You can fire up the console and see if everything is working. Depending on your distribution there should be a new icon in your Applications menu for VMware Server. On Red Hat that icon is located under Applications➪System Tools. Figure 2-9 shows what you are looking for.

Figure 2-10 shows the VMware Server Console on Linux.

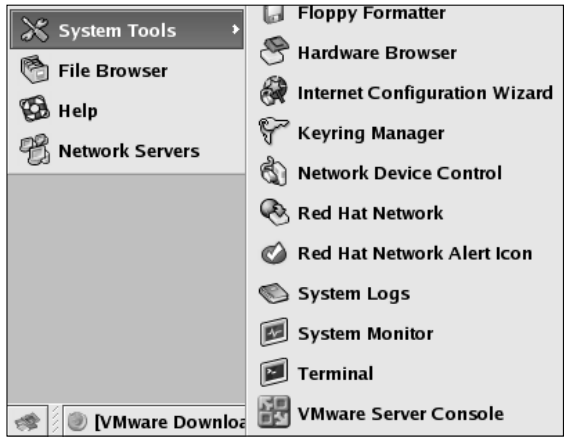

Figure 2-9

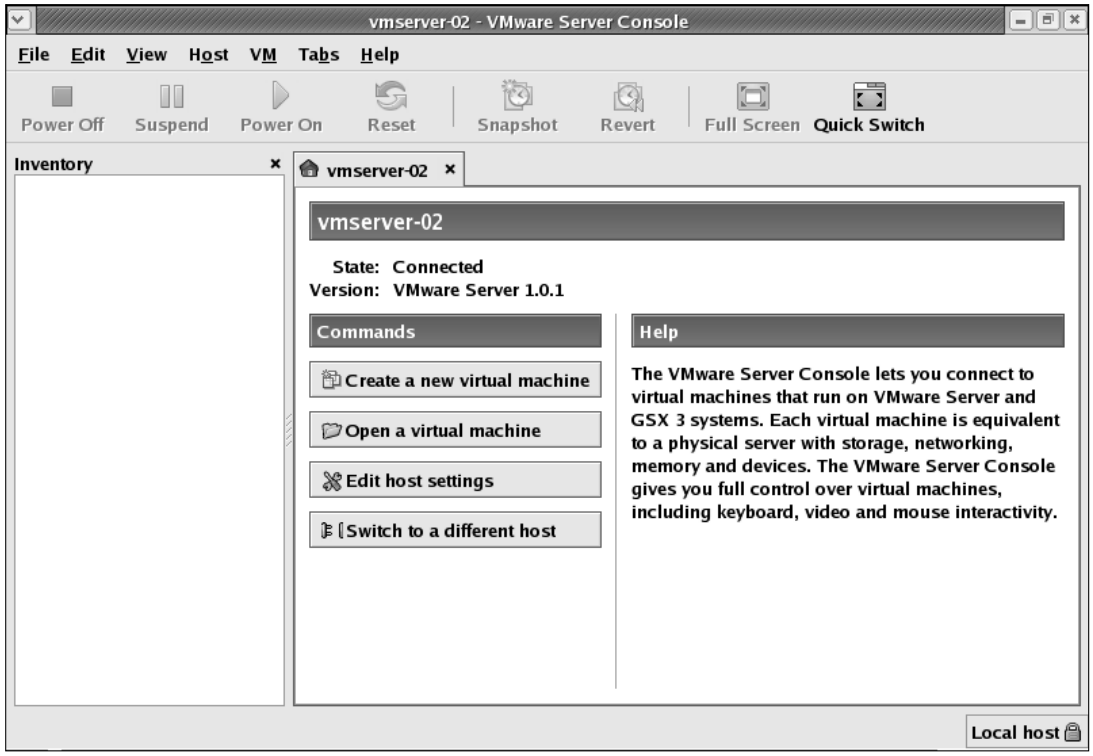
vmserver-02 - VMware Server Console
File Edit View Host VM Tabs Help
Power Off
Suspend
Power On
Reset
Snapshot
Revert
Full Screen
Quick Switch
Inventory
vmserver-02
vmserver-02
State: Connected
Version: VMware Server 1.0.1
Commands
Create a new virtual machine
Open a virtual machine
Edit host settings
Switch to a different host
Help
The VMware Server Console lets you connect to virtual machines that run on VMware Server and GSX 3 systems. Each virtual machine is equivalent to a physical server with storage, networking, memory and devices. The VMware Server Console gives you full control over virtual machines, including keyboard, video and mouse interactivity.
Local host

Figure 2-10

If you were unable to find any new icons in your Applications menu or on your desktop, refer to the "Launching the Server Console Manually" section in the troubleshooting material at the end of this chapter for more information.

If you upgrade your kernel you will need to re-run the configuration script in order to install modules compatible with your new kernel. Upgrade and reboot into your upgraded kernel. Open a console and type the following command to start the configuration script. Once complete you will have new modules matching your upgraded kernel.

```
su -

/usr/bin/vmware-config.pl
```

Install Overview

Now that you've completed the install on both Windows and Linux, I want to take a few moments to review what you've done:

- The Windows installer has taken care of installing VMware Server, VMware Server Console, VmPerl API, VmCOM API, C API, and the VMware Server Management Interface, if you selected it for install. You'll find this all neatly tucked away under `C:/Program Files/vmware`. If IIS was installed and you choose to install the web management interface, the installer has already taken care of creating the virtual directory in the IIS Management tool and setting it up for use.
- The Linux installer has taken care of installing VMware Server, VMware Server Console, and the VmPerl API. You still have the VMware Server Management Interface to install. That is covered in the next section.

You might be left with a few questions now that everything is coming together, but rest assured you will get to what all these pieces do and how to use them in the coming chapters. For now I want to get the Linux install complete.

Installing the VMware Server Management Interface on Linux

Okay, so while the Windows install already took care of this installation, you have to grab a different package from VMware to install the web management interface on Linux.

> *At this time the VMware Management Interface does not work on 64-bit Linux hosts. Please refer to the VMware web site at* `www.vmware.com` *for further information.*

Before jumping in you need to make sure your Linux host has a web server installed. More than likely, unless you have installed it already it does not.

1. On Red Hat you do this by accessing the Application➪System Settings ➪Add/Remove Applications menu.

2. Once the package management window is open, scroll down to the Servers section and place a checkmark beside Web Server. You do not need to manually change any of the default package selections for what you are doing here.
3. Click Update and have your CDs handy.

Once that is complete you can continue on.

> *Package management and installation on Linux varies greatly by distribution and the methods are too numerous to go through here. If you have trouble please consult your distribution's documentation on package management.*

Like before you need to obtain the binaries from the VMware web site at `www.vmware.com/download/server`. The binary you are interested in this time is the Linux Management Interface.

Once you have the archive downloaded, this is what you do:

1. Open a console and extract it into the current directory. For now assume you downloaded the archive into your home directory. Type the following command into your console:

```
tar zxf VMware-mui-<version>.tar.gz
```

> *Insert the version number you downloaded into the* `<version>` *tag.*

2. After a few moments the command will extract the contents of the tar indicated into a folder called `vmware-mui-distrib` in the same location as your archive.
3. The Management Interface install, like the others before, requires root access. Type the switch user command into your console. You will need to know the root password for your system.

```
su -
```

> *Once again be aware that once you switch users (*`su`*) to root, your current directory will change. You will need to change back into your home directory before continuing. You can do so with the following command:*

```
cd /home/<username>/vmware-mui-distrib
```

> *Remember, you cannot use the ~ modifier here because it refers to the root user's home directory now. If your distribution stores users' home directories in a different location, insert it in the preceding command in lieu of the one provided.*

4. Located inside the `vmware-mui-distrib` folder you'll find a file named `vmware-install.pl`. This Perl script, like the one used to install the server itself, will automate most of the install for you barring a few questions, of course.

```
./vmware-install.pl
```

> *The use of the* `./` *that precedes the command is required. If you receive a* `command not found` *error, verify you are in the correct directory and that you prefixed the command properly.*

5. Once the install starts, you will be asked a series of questions to help set up the management interface. Right off you'll have to read the End User License Agreement. Read and accept it to continue.

6. The installer will ask you where to place various files, libraries, and documentation manuals like before. The default paths provided should be used and considered accurate for your distribution. Unless you have a thorough understanding of your distribution or you are installing under some special circumstances, you should accept the defaults provided.

```
In which directory do you want to install the binary files?
[/usr/bin]

What is the directory that contains the init directories (rc0.d/ to rc6.d/)?
[/etc/rc.d]

What is the directory that contains the init scripts?
[/etc/rc.d/init.d]

In which directory do you want to install the VMware Management Interface files?
[/usr/lib/vmware-mui]

The path "/usr/lib/vmware-mui" does not exist currently.  This program is going to
create it, including needed parent directories.  Is this what you want? [yes]

In which directory would you like to install the documentation files?
[/usr/lib/vmware-mui/doc]

The path "/usr/lib/vmware-mui/doc" does not exist currently.  This program is going
to create it, including needed parent directories.  Is this what you want? [yes]
```

7. Now the installer wants to run the configuration script for you. This needs to be run at least once before the management interface will function properly. Go ahead and allow the install to run the configuration script.

```
Before running VMware Management Interface for the first time, you need to
configure it by invoking the following command:
"/usr/bin/vmware-config-mui.pl".  Do you want this program to invoke the command
for you now? [yes]
```

8. Set the number in minutes you will to allow before an HTTP session time out. The default is 60 minutes. If that seems a little long, feel free to edit that number to whatever you feel is appropriate. Enter your number or accept the default and continue.

```
Set the number of minutes before an http session times out.  (This is the length
of time before someone connecting to VMware Management Interface will be logged
out) [60]
```

If everything went well you should see an SSL certificate generated, `httpd.vmware` started, and a nice message that the installation of VMware Management Interface completed successfully.

It is possible at this point to receive an error when the script attempts to start the management interface. If you receive a message stating that the library file `libdb.so.3` is missing, refer to the section "The Case of the Missing Library: libdb.so.3" in the troubleshooting materials at the end of this chapter for more information.

Now you can open a browser and see what the management interface looks like. In order to do that, you need to understand a few things about the web management interface. The VMware Server Management Interface defaults to port 8222 on your web server, which is nothing more than a redirect to port 8333, which is the secure SSL site. No matter which port you browse to, you should arrive at the page shown in Figure 2-11. Open your browser and enter **http://localhost:8222**. If you know the name of your server, insert it instead of **localhost**.

The VMware Management Interface requires one of these browsers for proper operation:

- Internet Explorer 5.5 or 6.0
- Firefox 1.x
- Mozilla 1.x
- Netscape Navigator 7.0

Depending on your browser, you will receive a request to accept the SSL certificate. Because the certificate is generated by the installer, it is self-signed and will generate a warning that the signer is unknown. If you don't mind receiving the warning every time you access the site, that is fine or you can accept the certificate permanently. Details on how to do that will vary by browser, but you should be offered a choice to either accept the certificate permanently or install the certificate locally.

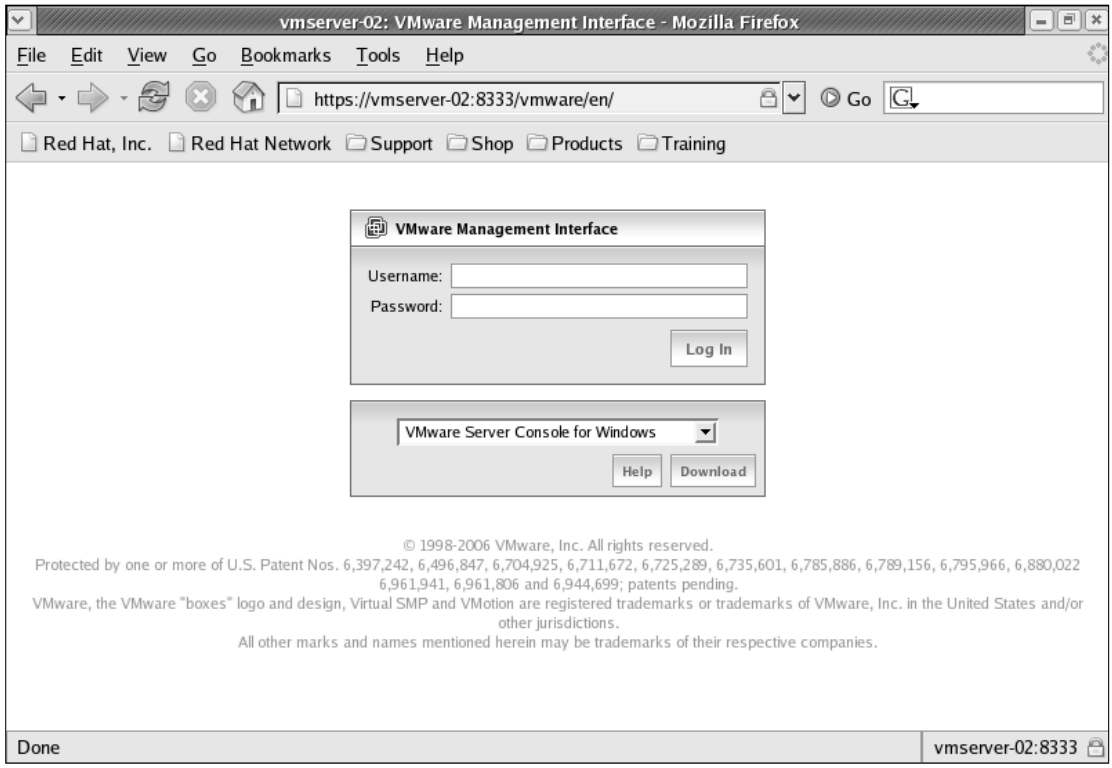
vmserver-02: VMware Management Interface - Mozilla Firefox
File Edit View Go Bookmarks Tools Help
https://vmserver-02:8333/vmware/en/
Go
Red Hat, Inc. Red Hat Network Support Shop Products Training
VMware Management Interface
Username:
Password:
Log In
VMware Server Console for Windows
Help
Download
© 1998-2006 VMware, Inc. All rights reserved.
Protected by one or more of U.S. Patent Nos. 6,397,242, 6,496,847, 6,704,925, 6,711,672, 6,725,289, 6,735,601, 6,785,886, 6,789,156, 6,795,966, 6,880,022 6,961,941, 6,961,806 and 6,944,699; patents pending.
VMware, the VMware "boxes" logo and design, Virtual SMP and VMotion are registered trademarks or trademarks of VMware, Inc. in the United States and/or other jurisdictions.
All other marks and names mentioned herein may be trademarks of their respective companies.
Done
vmserver-02:8333

Figure 2-11

If you have a firewall enabled on your Linux server, it will be necessary to allow port 8222/tcp and 8333/tcp through. See the "Enabling the MUI and Console Ports in Your Firewall" section in the troubleshooting material at the end of the chapter for more information.

Installing the Remote Client

The *remote client* is simply a machine other than the server on which you install the VMware Server Console. The remote client is also what allows your users to utilize your virtual machines housed on the server. As outlined previously in the chapter in "Remote Client Requirements," the hardware and software requirements for the remote client are considerably more lax than that of the server.

With that in mind you will install the remote client onto another workstation on your network. This workstation will later become one of your developer machines in Chapter 6. The next section covers the installation of the remote client on a Windows XP Professional SP2 workstation.

Installing the Remote Client on Windows

There are really two ways to accomplish this task, but the most common way to install the remote client is through the VMware Server Management Interface.

- One way is to download the VMware Server Windows client package from VMware. You can download the client package from the same location you downloaded the server binaries in the previous sections. The package is simply an archive that contains the Windows VMware Server Console application, the VmPerl API, and the VmCOM API. Each is contained in a separate executable within the main downloadable archive, so you can skip the API installs if you so choose using this method.
- The second method is via the VMware Server Management Interface you installed in the previous section. The catch with using this method is that only the VMware Server Console will be installed. The management interface does not offer the APIs as downloads. Your needs will determine which method you choose.

This section covers installing the remote client via the management interface:

1. Start by opening Internet Explorer on your Windows XP workstation. Once it is open, you want to browse to your host web management interface. You'll need to insert the name or IP address of your server. For this example you'll be using the web interface on `vmserver-03`, your Linux host. Remember, the interface runs on a nonstandard port so be sure to include that in your URL. In this example you will browse to `http://vmserver-01:8333` as shown in Figure 2-12.
    - If this page comes back not found, try to access it using the IP address of your host server. If it still fails and you have a Linux host, you may need to enable ports 8222/tcp and 8333/tcp in the host server's firewall if it is enabled. Refer to "Enabling the MUI and Console Ports in Your Firewall" section in the troubleshooting material at the end of the chapter for more information.
    - If you have trouble browsing to your server's web management page and you have a Windows-based host, try to access it by the server's IP address. The Windows Server 2003 firewall is not enabled by default; therefore, you will need to troubleshoot your network's name resolution if it continues to fail.

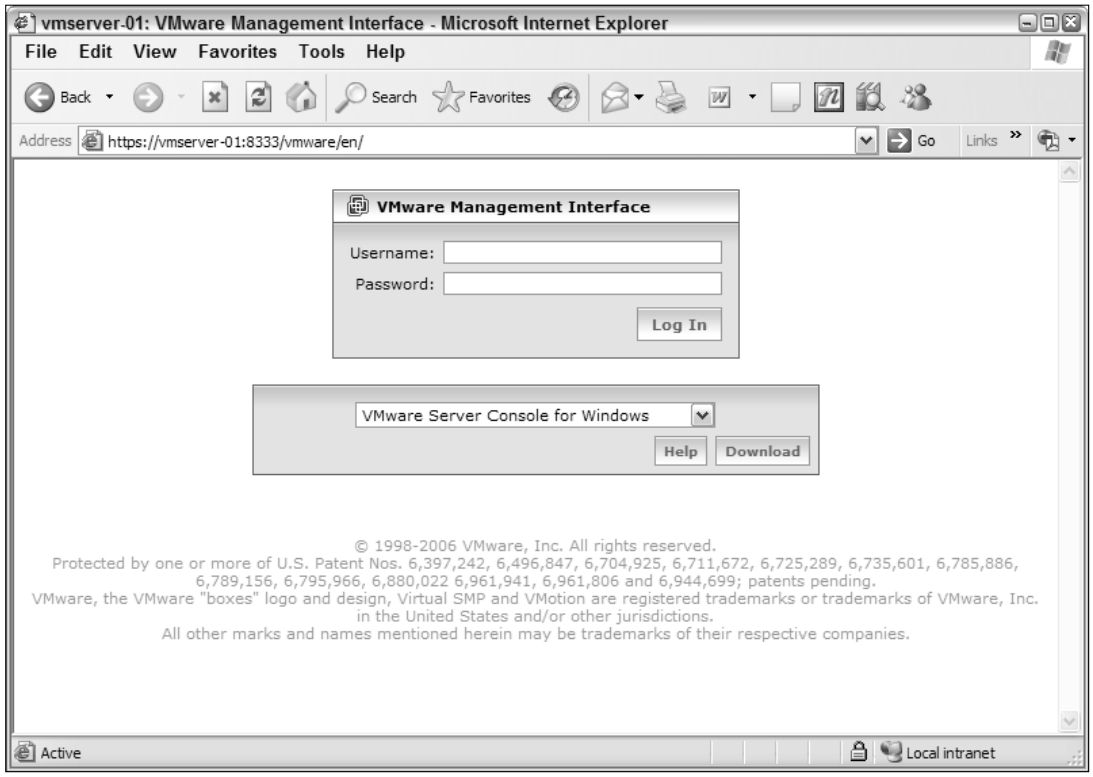
vmserver-01: VMware Management Interface - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites
Address https://vmserver-01:8333/vmware/en/
Go Links
VMware Management Interface
Username:
Password:
Log In
VMware Server Console for Windows
Help Download
© 1998-2006 VMware, Inc. All rights reserved.
Protected by one or more of U.S. Patent Nos. 6,397,242, 6,496,847, 6,704,925, 6,711,672, 6,725,289, 6,735,601, 6,785,886, 6,789,156, 6,795,966, 6,880,022 6,961,941, 6,961,806 and 6,944,699; patents pending.
VMware, the VMware "boxes" logo and design, Virtual SMP and VMotion are registered trademarks or trademarks of VMware, Inc. in the United States and/or other jurisdictions.
All other marks and names mentioned herein may be trademarks of their respective companies.
Active
Local intranet

Figure 2-12

2. If you look carefully at the center of the page, you'll notice a drop-down that says VMware Server Console for Windows with a Download button situated right below it. This is how the VMware Server Management Interface allows you to download the remote client to your workstation. If you go ahead and click the drop-down box, in addition to the Windows selection, you'll notice there are also selections for both a Linux tar.gz and rpm package for remote client installs on Linux workstations. I cover installing the remote client on a Linux workstation in the next section.

   Make sure the Windows package is selected and click Download. You will see the File Download screen pictured in Figure 2-13.

File Download - Security Warning
Do you want to run or save this file?
Name: VMware-console-1.0.0-27828.exe
Type: Application, 9.48 MB
From: vmserver-01
Run
Save
Cancel
While files from the Internet can be useful, this file type can potentially harm your computer. If you do not trust the source, do not run or save this software. What's the risk?

Figure 2-13

3. Now you can run it from its current location or download the package to the local machine for installation. Run it from its current location by choosing Run from the File Download box.

4. Once the download is complete, you will be presented with a security warning as shown in Figure 2-14. Go ahead and click Run.

Internet Explorer - Security Warning
Do you want to run this software?
Name: VMware-console-1.0.0-27828.exe
Publisher: VMware, Inc.
More options
Run
Don't Run
While files from the Internet can be useful, this file type can potentially harm your computer. Only run software from publishers you trust. What's the risk?

Figure 2-14

5. Next you will be presented with the installer, as shown in Figure 2-15.

VMware Server Console
Welcome to the installation wizard for VMware Server Console
The installation wizard will install VMware Server Console on your computer. To continue, click Next.
VMware Server
CONSOLE
WARNING: This program is protected by copyright law and international treaties.
< Back
Next >
Cancel

Figure 2-15

6. The rest is pretty self-explanatory, so accept the license agreement and default install directory and choose Install.

Once the installer is complete, you'll notice a new icon on your desktop for the VMware Server Console. Go ahead and fire it up and see what you have. It should be something similar to Figure 2-16.

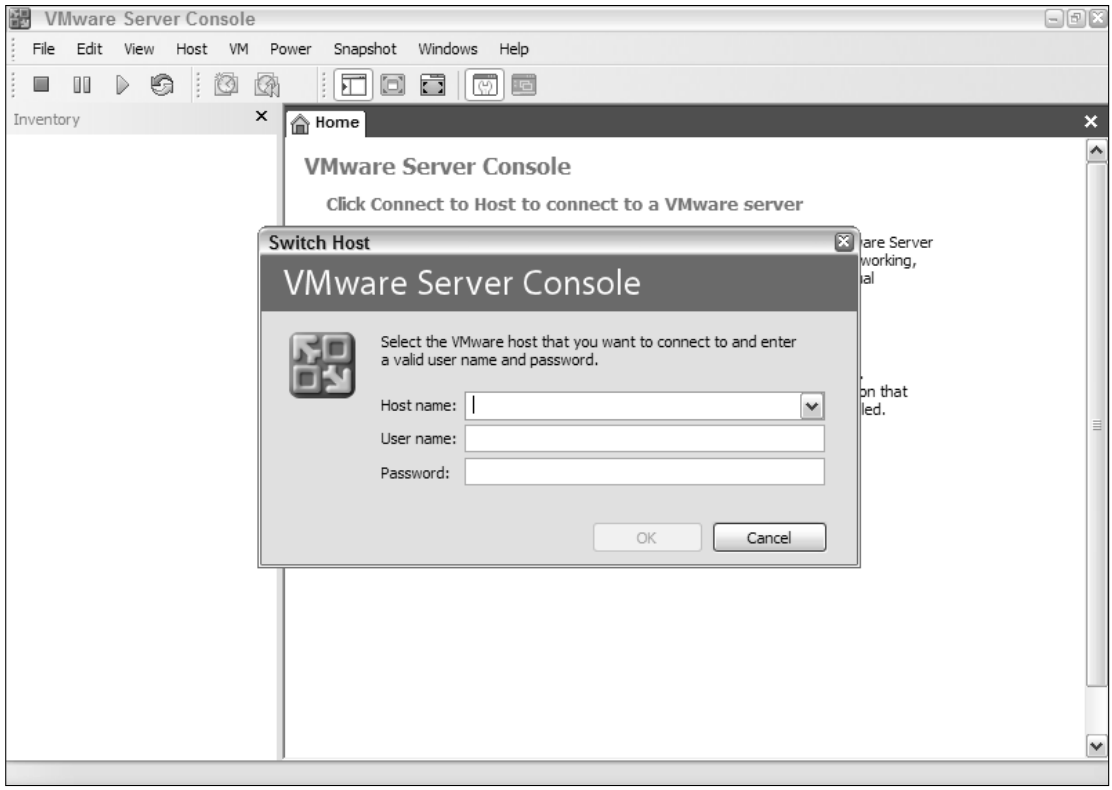
VMware Server Console
File Edit View Host VM Power Snapshot Windows Help
Inventory
Home
VMware Server Console
Click Connect to Host to connect to a VMware server
Switch Host
VMware Server Console
Select the VMware host that you want to connect to and enter a valid user name and password.
Host name:
User name:
Password:
OK
Cancel

Figure 2-16

Now you should notice a difference right off from what you saw when you launched the server console on your host. Because the console on the host is local it doesn't prompt you for a server to connect to; instead it just connects to the localhost. Here your remote client needs to know where your host server is. Enter the server name, your user name on the host, and your password, and click OK. An example is shown in Figure 2-17.

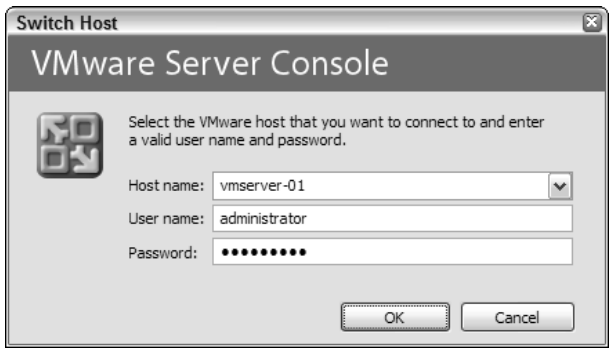
Switch Host
VMware Server Console
Select the VMware host that you want to connect to and enter a valid user name and password.
Host name: vmserver-01
User name: administrator
Password: •••••••••
OK
Cancel

Figure 2-17

If you installed VMware Server on a nonstandard port (not port 902) enter the host name as host:port.

If everything works out, after a few moments the server console should load with the name of your host computer in the title bar and the current inventory of virtual machines present in the left pane. Your server doesn't have any virtual machines yet, so your inventory is empty. You can compare your results with Figure 2-18 if you are unsure.

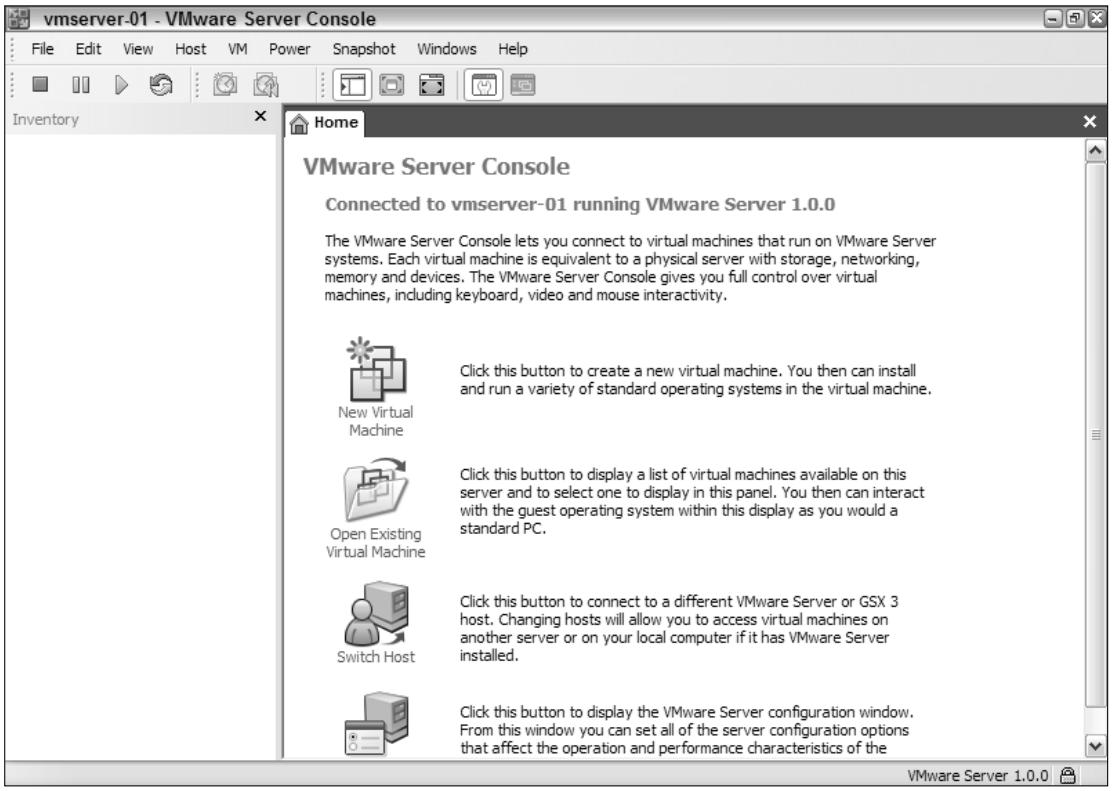
vmserver-01 - VMware Server Console
File Edit View Host VM Power Snapshot Windows Help
Inventory
Home
VMware Server Console
Connected to vmserver-01 running VMware Server 1.0.0
The VMware Server Console lets you connect to virtual machines that run on VMware Server systems. Each virtual machine is equivalent to a physical server with storage, networking, memory and devices. The VMware Server Console gives you full control over virtual machines, including keyboard, video and mouse interactivity.
New Virtual Machine
Click this button to create a new virtual machine. You then can install and run a variety of standard operating systems in the virtual machine.
Open Existing Virtual Machine
Click this button to display a list of virtual machines available on this server and to select one to display in this panel. You then can interact with the guest operating system within this display as you would a standard PC.
Switch Host
Click this button to connect to a different VMware Server or GSX 3 host. Changing hosts will allow you to access virtual machines on another server or on your local computer if it has VMware Server installed.
Click this button to display the VMware Server configuration window. From this window you can set all of the server configuration options that affect the operation and performance characteristics of the
VMware Server 1.0.0

Figure 2-18

That's it. The VMware Server Console is now installed on your Windows workstation. From here you can create new virtual machines on the server, open existing ones, and configure the host and client application settings. I cover more of what you can do with the server console in Chapter 3.

Because I skipped over installing the APIs by using the management interface method, I want to talk about it briefly before moving on. If you want to install the APIs locally on the remote client, you will need to download the client installer package mentioned earlier in this section. Once you have the installer, you can unpack it and install VmPerl, VmCOM, or both. In most situations, however, you just need to install the console, and the web management interface install method can't be beat for simplicity.

Installing the Remote Client on Linux

The previous section covered installing the remote client on a Windows XP workstation. I want to complete the circle now by covering a remote client install on a Linux workstation. In this section I am going to use a workstation running Fedora Core 5 with the stock 2.6.15-1.2054_FC5smp kernel.

As I touched upon before, there is more than one way to accomplish this:

- If you want the full package including the VmPerl API for your remote workstation, the best method is to download the VMware Server Linux client package from `www.vmware.com/download/server`. Unlike the previous Linux packages, however, this one includes both the tar and rpm versions of the install as well as the VmPerl API installer in tar format.
- The second method is to install via the web management interface. This offers by far the simplest method of distributing the console to remote workstations but has the limitation of no APIs being available for installation. More than likely you will not be installing the APIs on more than a couple of workstations, if any at all.

Again, the following steps walk you through the method that uses the management interface:

1. First start by opening a browser on your remote Linux workstation and navigating to your host. For this example again you'll be using the web interface on `vmserver-03`, your Linux host. Remember, the interface runs on a nonstandard port so be sure to include that in your URL. For this example, browse to `http://vmserver-02:8333`, as shown in Figure 2-19.

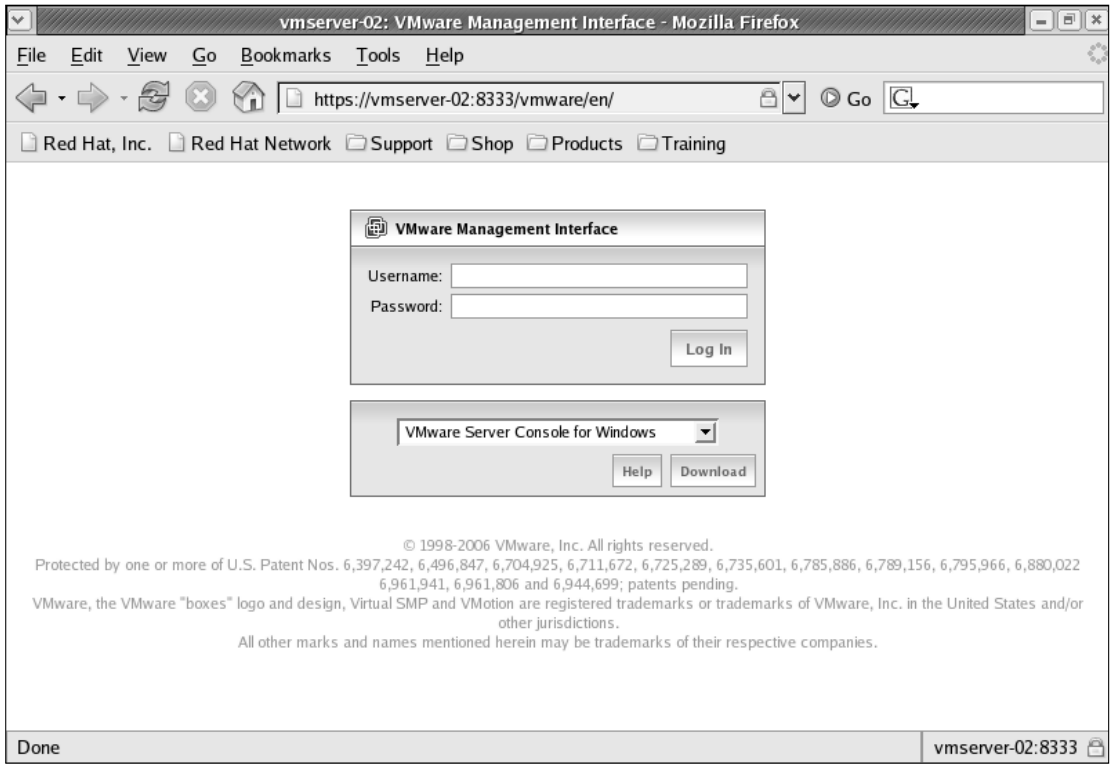
vmserver-02: VMware Management Interface - Mozilla Firefox
File Edit View Go Bookmarks Tools Help
https://vmserver-02:8333/vmware/en/
Go
Red Hat, Inc. Red Hat Network Support Shop Products Training
VMware Management Interface
Username:
Password:
Log In
VMware Server Console for Windows
Help
Download
© 1998-2006 VMware, Inc. All rights reserved.
Protected by one or more of U.S. Patent Nos. 6,397,242, 6,496,847, 6,704,925, 6,711,672, 6,725,289, 6,735,601, 6,785,886, 6,789,156, 6,795,966, 6,880,022 6,961,941, 6,961,806 and 6,944,699; patents pending.
VMware, the VMware "boxes" logo and design, Virtual SMP and VMotion are registered trademarks or trademarks of VMware, Inc. in the United States and/or other jurisdictions.
All other marks and names mentioned herein may be trademarks of their respective companies.
Done
vmserver-02:8333

Figure 2-19

- If this page comes back not found, try to access it using the IP address of your host server. If it still fails and you have a Linux host, you may need to enable ports 8222/tcp and 8333/tcp in the host server's firewall if it is enabled. Refer to the "Enabling the MUI and Console Ports in Your Firewall" section in the troubleshooting material at the end of the chapter for more information.
- If you have trouble browsing to your server's web management page and you have a Windows-based host, try to access it by the server's IP address. The Windows Server 2003 firewall is not enabled by default; therefore, you will need to troubleshoot your network's name resolution if it continues to fail.

2. The center of the web management page offers the VMware Server Console as a download. Click the drop-down box and select either the rpm or tar.gz version for Linux. (I'm going to stick with the tar.gz version because it is compatible with the most distributions.) Select VMware Server Console for Linux (tar.gz) from the selection box and click the Download button.
3. You should be prompted for what to do with the download as shown in Figure 2-20. Choose Save to Disk and click OK. Depending on your distribution and browser, the file may download to your desktop or your home directory. In this example the download is saved to your home directory.

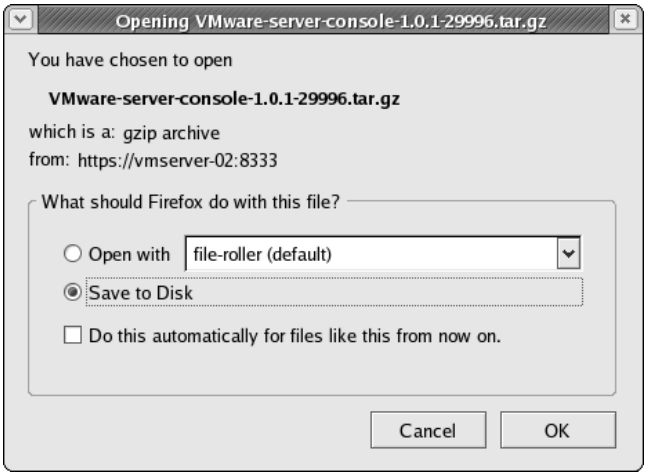

Figure 2-20

4. Open a console and browse to the directory the file downloaded to; in this case it's your home directory.
5. You need to extract the tar archive so you can get the install rolling. Type the following in your console:

```
tar zxf VMware-console-<version>.tar.gz
```

Insert the version number you downloaded into the `<version>` *tag.*

6. After a few moments the command will extract the contents of the archive into a folder called `vmware-console-distrib` in the same location as the archive itself.

7. So, as with the Linux installs previously, there's a Perl script you need to run, and you must have root access to do it. Issue the switch user command to get things ready. Remember, you will need to know the root password for the workstation you are installing the console on.

```
su -
```

After you issue the switch user (`su`) command your current directory will change. You will need to change directories back to where you extracted the archive before continuing. Assuming the archive is in your home directory you can issue the following command, inserting your user name in place of the `<username>` tag:

```
cd /home/<username>/vmware-console-distrib
```

8. Start the install process by typing the command that follows into your console. Remember you must prefix the filename with `./`:

```
./vmware-install.pl
```

9. You'll be presented with the license agreement first. Once you accept it you can move on.

10. This install should be relatively simple compared to the ones you already tackled. First, the installer has to figure out where you want things. Just accept the default locations and allow the installer to create the necessary directories.

```
In which directory do you want to install the binary files?
[/usr/bin]

In which directory do you want to install the library files?
[/usr/lib/vmware-console]

The path "/usr/lib/vmware-console" does not exist currently.  This program is going
to create it, including needed parent directories.  Is this what you want? [yes]

In which directory do you want to install the manual files?
[/usr/share/man]

In which directory do you want to install the documentation files?
[/usr/share/doc/vmware-console]

The path "/usr/share/doc/vmware-console" does not exist currently.  This program is
going to create it, including needed parent directories.  Is this what you want?
[yes]
```

11. Now, as in the past, the installer wants to run the configuration script for you. This script must be run at least once before the console will operate properly. Go ahead and let the installer run the configuration.

```
Before running VMware Server Console for the first time, you need to configure it
by invoking the following command: "/usr/bin/vmware-config-console.pl".  Do you
want this program to invoke the command for you now? [yes]
```

12. The question that comes next is important. Think back to your server install and remember, did you use the standard port of 902 or did you have to use a nonstandard port, like 904? Enter the port number you used during the server install now.

```
What port do you want the remote console to use to connect to the server (Only
change this if you have not used the default value during server installation)?
[902]
```

Unlike the Windows remote client where you must refer to it as `host:port` *if the host uses a nonstandard port, the Linux remote client does not require it. If you use a nonstandard port on your host, enter it here, and the Linux remote client will default to looking at the new port number. Bear in mind, however, that this is the new default for that remote client. If it connects to any other VMware Server hosts that run on the standard port of 902, you will have to refer to them as* `host:902`.

If all is well, you should receive a nice VMware Server Console install completed successfully message.

Now that the remote client install is complete, fire it up to see what you have and to make sure you can connect to your server. Depending on your distribution you should have an icon in your Applications menu to start the server console. You should see something similar to Figure 2-21.

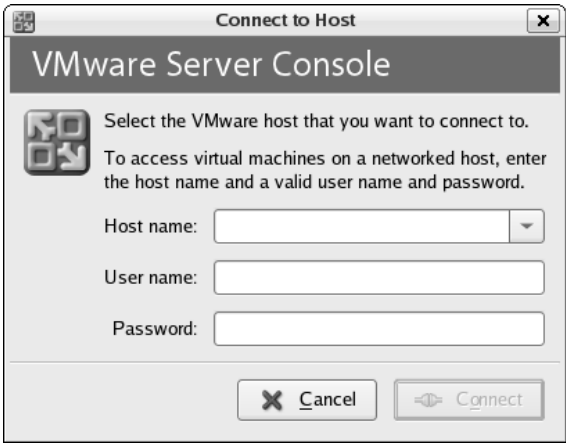
Connect to Host
VMware Server Console
Select the VMware host that you want to connect to.
To access virtual machines on a networked host, enter the host name and a valid user name and password.
Host name:
User name:
Password:
Cancel
Connect

Figure 2-21

If you were unable to find any new icons in your Applications menu or on your desktop, refer to the "Launching the Server Console Manually" section in the troubleshooting discussion at the end of this chapter for more information.

Now, because this is a remote client, you need to tell it what server to connect to. For example, to connect to the Linux host you installed the client from you would enter the information as shown in Figure 2-22.

Connect to Host
VMware Server Console
Select the VMware host that you want to connect to.
To access virtual machines on a networked host, enter the host name and a valid user name and password.
Host name:
vmserver-02
User name:
eric
Password:
Cancel
Connect

Figure 2-22

If you installed VMware Server on a nonstandard port, enter the host name as `host:port` *unless you entered the port during the configuration.*

After a few moments the server console should load with the name of your host computer in the title bar and the current inventory of virtual machines present in the left pane. Your server doesn't have any virtual machines yet so the inventory is empty. You can compare your results with Figure 2-23 just to make sure.

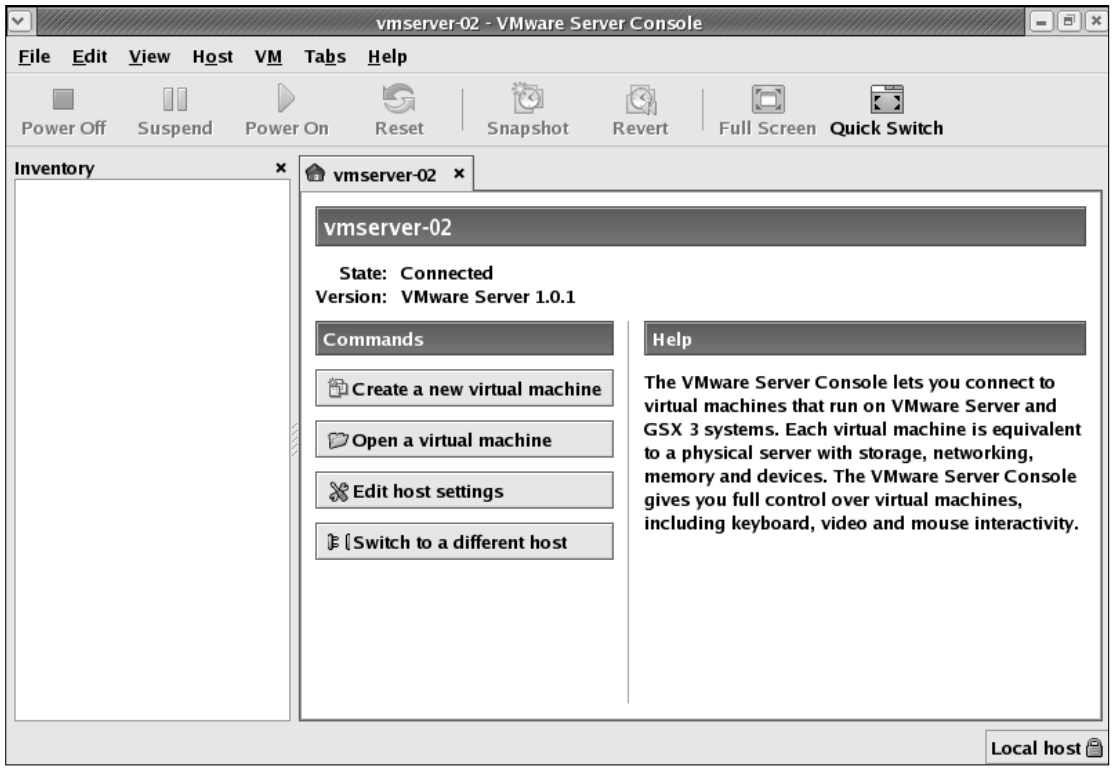
vmserver-02 - VMware Server Console
File Edit View Host VM Tabs Help
Power Off
Suspend
Power On
Reset
Snapshot
Revert
Full Screen
Quick Switch
Inventory
vmserver-02
vmserver-02
State: Connected
Version: VMware Server 1.0.1
Commands
Create a new virtual machine
Open a virtual machine
Edit host settings
Switch to a different host
Help
The VMware Server Console lets you connect to virtual machines that run on VMware Server and GSX 3 systems. Each virtual machine is equivalent to a physical server with storage, networking, memory and devices. The VMware Server Console gives you full control over virtual machines, including keyboard, video and mouse interactivity.
Local host

Figure 2-23

The VMware Server Console is now installed on your Linux workstation. From here you can create new virtual machines on the server, open existing ones, and configure the host and client application settings. I cover more of what you can do with the server console in Chapter 3.

Troubleshooting the Install

Things go wrong — that's life — but when you are setting up VMware Server, you have a little more control over it. A few pitfalls can arise through the install process, so this section of the chapter covers some of the more common ones.

Having to Compile Your Own Kernel Modules

This section helps you navigate compiling your own kernel modules during the install sequence for VMware Server. If you saw the following text during the install, you're in the right place:

```
Trying to find a suitable vmmon module for your running kernel.

None of the pre-built vmmon modules for VMware Server is suitable for your running
kernel.  Do you want this program to try to build the vmmon module for your system
(you need to have a C compiler installed on your system)? [yes]
```

Now this can get a little tricky, so hang on.

The following procedure covers most of Red Hat's newer distributions. Due to the vast number of ways to install packages on various Linux distributions, a single method had to be singled out. Most of what follows will work in all distributions except the use of `yum`*, the Red Hat–specific package installer.*

1. The first requirement needed to build your own modules is a C compiler. The good news is that almost all distributions install a C compiler by default. Go ahead and say `yes` to the preceding message, and you should see the following text if the install script finds your compiler:

```
Using compiler "/usr/bin/gcc".  Use environment variable CC to override.
```

That's a good message. The install script found your compiler and is ready for you to move on.

If you are using SuSE Enterprise Linux 8 as your host operating system, the `gcc` *compiler is not installed by default. You can install the* `gcc` *compiler package by running YaST and selecting the C++ Compiler and Tools package for installation.*

2. The second requirement is that you have the C header files installed that match the currently running kernel. This is where you can run into problems because some distributions do not include the kernel headers by default. First look at what it is asking you for:

```
What is the location of the directory of C header files that match your running
kernel? [/usr/src/linux/include]
```

3. Go ahead and accept the default directory and press Enter.

- If the install continues, you're the lucky one and are using a distribution that sets up a link between `/usr/src/linux/include` and the directory where your headers are located. Return to the appropriate installation section of this chapter and complete the steps.
- More than likely, however, the installer cannot find your header files, and you need to install them. You can be sure of this if you receive the following message:

```
The path "/usr/src/linux/include" is not an existing directory.
```

If this message or one like it is the message you receive, continue with the steps in this section.

4. It's not a hard problem to fix. You just have to install them before you can continue. Leave the current console with the installer running up — you'll be coming back to it — and open a new console window. Type the following command to become root:

```
su -
```

5. Now, you need to determine what kind of kernel the host is running, single processor or SMP (multiprocessor). Type the next command and take note of the result:

```
uname -r
```

It will return the current running kernel version. Using my Linux host as an example, the output is `2.6.9-34.EL`. This tells you that you are running a single processor kernel. If the output were `2.6.9.EL-smp`, you would be running an SMP, or multiprocessor, kernel. This output determines how you will shape the next command.

6. Type the following in your console for a single processor kernel:

```
yum install kernel-devel
```

Or if you have the SMP kernel, type this instead:

```
yum install kernel-smp-devel
```

This will download and install the headers for your running kernel. There may be some dependencies added to the update by `yum`. This is normal. Accept any additional packages the package installer wants to download.

7. So, you have your kernel headers, but where are they? Red Hat by default places the headers in the `/lib/modules/<kernel version>/build/include` directory. Type the following command in the console to make sure everything is where it's supposed to be:

```
cd /lib/modules/$(uname -r)/build/include
```

The `$(uname -r)` is simply a shortcut method of including your kernel version on the command line. It's usually easier than typing `2.6.9.EL` although that will work as well.

8. You should at this point change into the `include` directory. You can verify this by typing `pwd`, the print working directory command, into the console and checking the path it returns.
9. Write down the path and switch back to your install window. Enter that path for the location of your C header files and press Enter. The installer should now continue with the build.

If you do not happen to be running a flavor of Red Hat on your host server, most but not all of the instructions in this section will still apply. `yum`, the package installer used in the preceding steps to download the kernel headers, is specific to Red Hat–based distributions. Other distributions use other programs — for example, SuSE uses something called YaST, and Ubuntu uses a program called `apt`. Consult your distribution's documentation for the proper way to download and install the kernel header package and the location to which it installs.

Verifying Port 902 Is Free

Port 902 is the default port used for remote clients to connect to the host server via the VMware Server Console. If you're reading this section, it is most likely because the configuration script during the VMware Server install said that port 902 was not free. Chances are, however, that it is free and safe for use, that you can just override the suggested port with 902, and that everything will work fine. Typically this happens when `xinetd` is not installed on the server and the install script relies on the `/etc/services` to determine if something is using port 902.

Luckily it's easy enough to determine if something is listening on port 902/tcp. Open a console and type the following command:

```
netstat -ntl
```

So you have an idea of what this command is doing, the `netstat` command is used to print Internet connections. The arguments used here are don't resolve host names (`n`), a socket type of tcp (`t`), and display listening server sockets (`l`).

The preceding command will provide a list of listening ports on your server similar to the one provided in the code block that follows:

```
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address               Foreign Address  State
tcp        0      0 0.0.0.0:111                 0.0.0.0:*        LISTEN
tcp        0      0 127.0.0.1:50000             0.0.0.0:*        LISTEN
tcp        0      0 127.0.0.1:50002             0.0.0.0:*        LISTEN
tcp        0      0 0.0.0.0:33811               0.0.0.0:*        LISTEN
tcp        0      0 127.0.0.1:631               0.0.0.0:*        LISTEN
tcp        0      0 127.0.0.1:25                0.0.0.0:*        LISTEN
tcp        0      0 :::22                       :::*             LISTEN
```

Under the local address column you will see a list of IP addresses, followed by a colon, followed by a port number. If none of the addresses list 902 as a port number, then it is free to use for your console. The chances are slim that it will be in use prior to the VMware Server installation; however, it is possible.

Installing xinetd

`xinetd` is an Internet services daemon. It simply controls various Internet services. Now, when I say Internet services I mean a service that listens on a specific port or ports to provide a service, like mail delivery. The Internet services daemon provides management for these services to ensure their automatic startup and port assignments. Because the VMware Server Console listens for connections on port 902/tcp it requires the use of `xinetd`. Some distributions do not come standard with `xinetd` installed. Most notably the Fedora line from Red Hat has a tendency to not install this daemon by default. Here's how to install `xinetd` on Red Hat.

> **/etc/services**
>
> The `/etc/services` file contains a list of standard services and their port numbers as defined by RFC 1700, "Assigned Numbers." This enables clients and servers to map service names to TCP and UDP port numbers. This is a list of standard services and not necessarily what is actually running on your server.

Even though I use the Fedora line from Red Hat as an example of a distribution that does not install `xinetd` *by default it is in no way supported as a host operating system for VMware Server.*

1. Open a console and `su` to root:

```
su -
```

2. Use the Red Hat package installer `yum` to download and install the package:

```
yum install xinetd
```

3. Once the download is complete and the package is installed, you should see something similar to what follows:

```
Installed: xinetd.i386 <version number>
Complete!
```

That's all there is to it.

Now, because you just installed `xinetd`, that means the configuration script during the VMware Server install probably failed. Using the same console make sure you are still root and execute the following command:

```
/usr/bin/vmware-config.pl
```

You'll see the installer stop the various VMware services and put you back into the configuration script. Find your place in the installation procedure earlier in this chapter and continue.

As the configuration script stops the various services, you will notice that stopping `xinetd` *fails. This is normal because you just installed the service, and it was never started. Some distributions may, however, start the service as a function of the package installer.*

The Case of the Missing Library: libdb.so.3

It is possible that when the system attempts to start up the management interface you could receive an error similar to the following one:

```
Couldn't find necessary components on your system.  It appears that you are missing
the following library: libdb.so.3.

Some Linux distributions are known to ship without these libraries.  From your
Linux distribution CD, install this RPM package: compat-db-<#>.<#>.<##>-
```

```
<#>.i386.rpm or libdb#.deb, where <#> is a version number particular to your
version of the distribution.

If your distribution CD does not have this package, contact your vendor for a
suitable library.

If you install this package after you installed the management interface software,
start the management interface's Apache server with the following command:
/etc/init.d/httpd.vmware start
```

The library being referred to is a compatibility library for Berkeley database compatibility. On Red Hat the fix is simple enough using your package manager `yum`:

1. Open a console and `su` to root as before:

```
su -
```

2. Next you want to fire up `yum` and download the `compat-db` package. Enter the following into your console:

```
yum install compat-db
```

3. Once complete, restart the VMware Management Interface with the following command:

```
/etc/init.d/httpd.vmware start
```

The server should start up now without incident.

Other distributions may package this library differently. Refer to your distribution's documentation on package management and installation for assistance.

Launching the Server Console Manually

With some Linux distributions the install can and will install an application launcher into one of your Application menus; on Red Hat it is installed into the Applications➪System Tools menu. However, with some distributions the install cannot, for whatever reason, insert an application launcher icon for you. An example of this is Fedora Core 5. In these cases you will need to either launch the server console manually or create your own application launcher.

- VMware Server installs all of its executable binaries into the `/usr/bin` directory. To manually launch the server console, open a command prompt and type the following:

```
/usr/bin/vmware-server-console
```

- Now, because it is a hassle to open a console and type that every time you want to launch the server console, you can create an application launcher for the desktop.

  In most Linux distributions if you right-click somewhere on your desktop a context menu will appear. Look for an item in the menu that says "Create Launcher," or something similar. Figure 2-24 shows the context menu using the Fedora Core 5 workstation as an example.

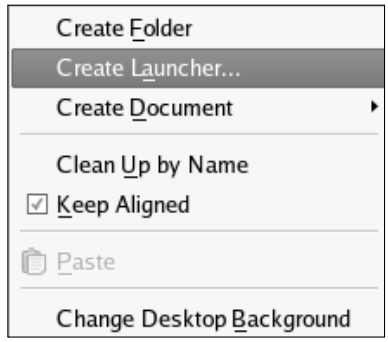

Figure 2-24

Once you select Create Launcher, you should see a screen similar to Figure 2-25, only without anything entered. Enter the text shown in Figure 2-25 and click OK. The VMware Server icon shown in the figure is installed into `/usr/share/pixmaps` by the installer. Click the icon button and scroll down to choose the `console.png` icon.

> *Some versions of the client installer do not install the console icon correctly. If you cannot locate the* `console.png` *icon under* `/usr/share/pixmaps`*, you can find it inside the extracted installer directory under* `/lib/share/pixmaps`*. This is the directory you installed the remote console from. You will have to copy that icon into your* `/usr/share/pixmaps` *folder as root in order for it to work seamlessly.*

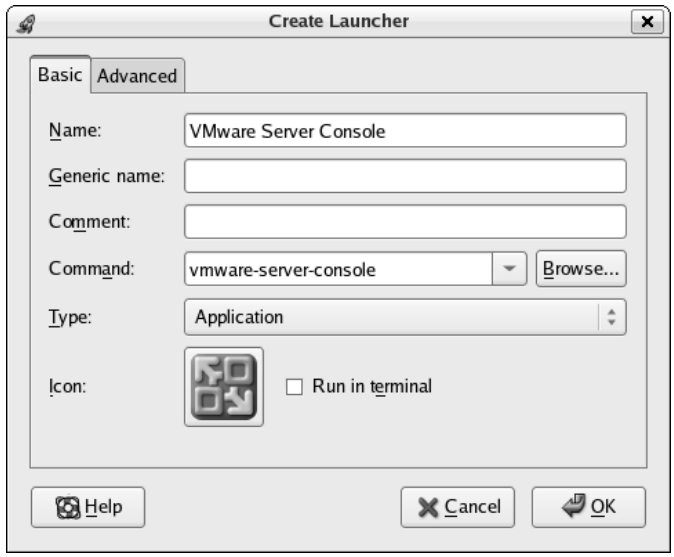

Figure 2-25

The actual screen may differ per distribution and X windows manager, but the concept is the same. Instructions on adding the icon to your Applications menu varies. Consult your distribution's documentation for information on how to add application launchers to your menu.

Enabling the MUI and Console Ports in Your Firewall

Most Linux distributions today ship standard with a firewall built in, and they usually enable it right out of the box. This forces you to manually add some rules to allow remote clients to connect to your Linux host's server console and remote web management interface. As previously indicated, the ports needed in a standard VMware Server install with the VMware Server Management Interface are as follows:

- **Port 902/tcp** — Used for remote client communication with the host server
- **Port 8222/tcp** — Used for web access to the host machine's web management interface
- **Port 8333/tcp** — Used for secure SSL web access to the host machine's web management interface

Sticking with the example of using Red Hat, you can enable these ports in Red Hat Enterprise Linux 4 WS, your Linux host server, as follows:

1. Red Hat, and most other Linux distributions, provides a GUI configuration tool to modify the firewall rules. On Red Hat this tool is called `system-config-securitylevel`. You can find this in the Applications➪System Settings menu as Security Level or by typing `system-configsecuritylevel` at the console. Go ahead and start it and enter the root password. You should see the window shown in Figure 2-26.

Figure 2-26

2. So you don't check anything that might cause you trouble later, take a quick look at what's available on this screen:

   - First is a list of common **Trusted services**. If you checked WWW (HTTP), port 80/tcp would be enabled, SSH would enable port 22/tcp, and so on.
   - Below that are **Trusted devices**. If you selected eth0 as a trusted device, then all traffic coming from that network adapter would be let though unharmed.
   - Neither of the first two bullets is what you want, or too much of what you want. The **Other ports** section is where you need to enter your information. Each port here needs to be entered as *port:socket*, for example, 80:tcp. Multiple entries in this field should be separated by commas. Type the following into the box to enable the ports you need and click OK. It should look like Figure 2-27 once complete.

```
902:tcp, 8222:tcp, 8333:tcp
```

If you use a nonstandard port for console communications, enter it instead of port `902:tcp`.

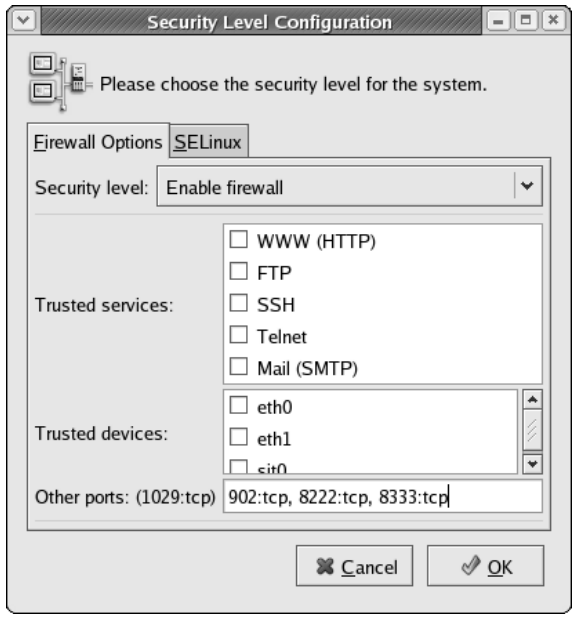

Figure 2-27

The results are instantaneous. Once you open the ports on this screen and click OK, switch to your remote client and test it out. You should now be able to connect via the remote console client and open the web management interface on the host server.

Other distributions offer similar interfaces to modify the firewall policy for the server. If you need assistance, please refer to the documentation for that server's firewall configuration.

Summary

As you can see VMware offers you many options when it comes to installation and remote client platforms. This chapter covered the following:

- Host hardware and operating system requirements
- Remote client hardware and operating system requirements
- VMware Server installation on Windows and Linux
- Remote server installation on Windows and Linux
- Installation troubleshooting

As you progressed through each type of install, things were set up and configured behind the scenes that you have not even become aware of yet. The next chapter dives into the layout of the server console, the host and guest application settings available, virtual hardware, and virtual networking and covers some of the useful command-line utilities available to you.

3

VMware Configuration Nuts and Bolts

So, now that I have covered the installation procedure and you have no doubt started to tinker around with the console and various settings, I want to take the time to give you a basic walk-through of a few items that will be of value to you in the coming chapters.

No question that the server console will be where you spend a majority of your time. It handles essentially all aspects of your server from configuration of the host to the creation and utilization of your guests. With that in mind I think it is only proper to take a few minutes to go over the console interface and its settings.

Getting to Know the Server Console

Like any software application the interface is the key to its utilization. VMware Server provides a very robust interface that not only allows you to configure and monitor your host server, but also provides a window into the guest virtual machines you house. This interface, as you will see in the following sections, is comprised of several different parts.

I want to start by refreshing your memory on the server console as a whole. Figure 3-1 shows the server console. Notice I have added a few virtual machines into the inventory for use during this chapter.

I cover just how to add virtual machines into inventory in Chapter 6.

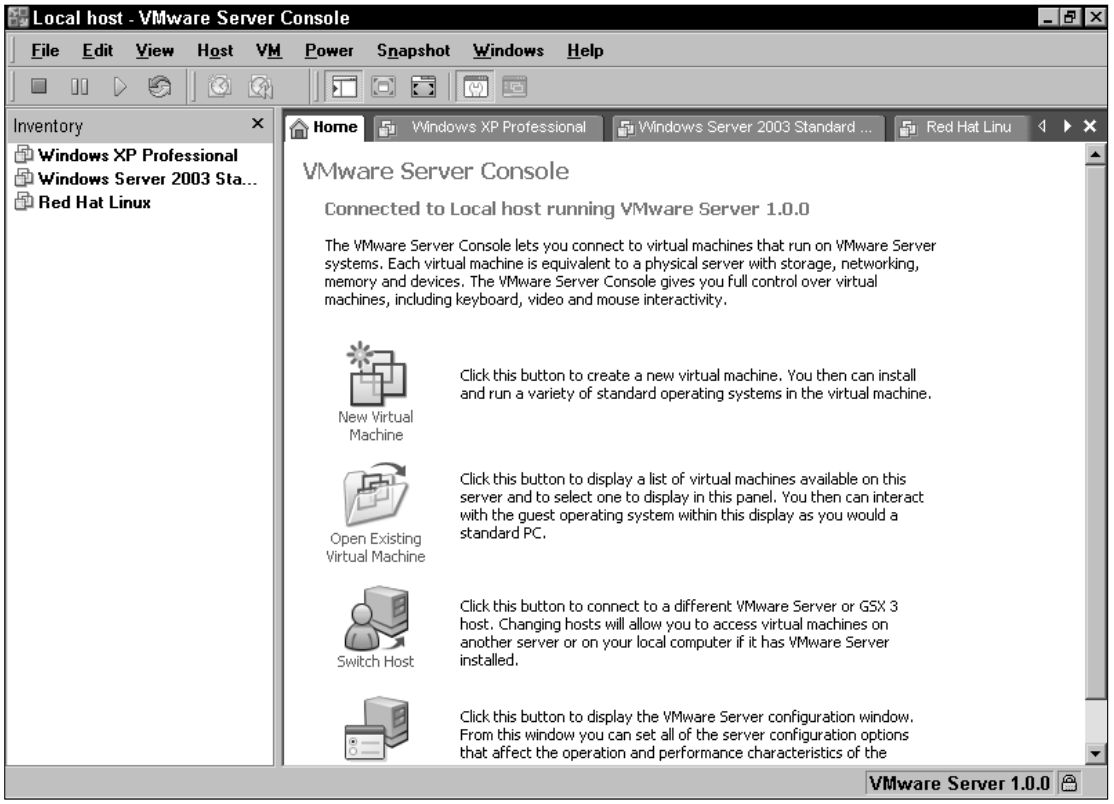
Local host - VMware Server Console
File Edit View Host VM Power Snapshot Windows Help
Inventory
Windows XP Professional
Windows Server 2003 Sta...
Red Hat Linux
Home
Windows XP Professional
Windows Server 2003 Standard ...
Red Hat Linu
VMware Server Console
Connected to Local host running VMware Server 1.0.0
The VMware Server Console lets you connect to virtual machines that run on VMware Server systems. Each virtual machine is equivalent to a physical server with storage, networking, memory and devices. The VMware Server Console gives you full control over virtual machines, including keyboard, video and mouse interactivity.
New Virtual Machine
Click this button to create a new virtual machine. You then can install and run a variety of standard operating systems in the virtual machine.
Open Existing Virtual Machine
Click this button to display a list of virtual machines available on this server and to select one to display in this panel. You then can interact with the guest operating system within this display as you would a standard PC.
Switch Host
Click this button to connect to a different VMware Server or GSX 3 host. Changing hosts will allow you to access virtual machines on another server or on your local computer if it has VMware Server installed.
Click this button to display the VMware Server configuration window. From this window you can set all of the server configuration options that affect the operation and performance characteristics of the
VMware Server 1.0.0

Figure 3-1

Since I mentioned the inventory, it is as good a place as any to start. The *Inventory* pane, as shown in Figure 3-2, gives you a table of contents, if you will, of the virtual machines on the host server. Now, as you will see in Chapter 6, these are not necessarily the only virtual machines on the server. This is simply a list of the guests that have been configured for remote clients to use. Each entry in the inventory has context-sensitive menus that allow for various things like powering on and off the machine. As you will learn, most commands that can be accessed from the toolbar or menu can be activated via the context menus present in the Inventory pane as well.

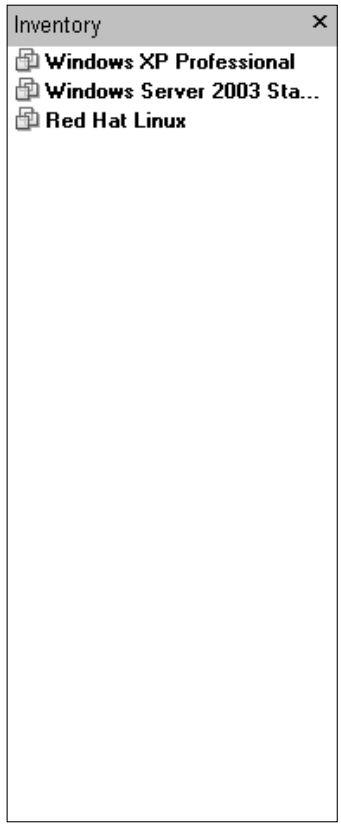

Figure 3-2

Now, it wouldn't be much fun to wade around context menus all day trying to get everything done. That is why the inventory pane I described previously is connected directly to what is displayed in the main pane. The *main* pane is where everything happens. You will create, configure, and use your virtual machines through their respective tabs in the main pane. For example, if you choose the Windows XP Professional guest in the Inventory pane, its corresponding tab will be shown in main pane. Figure 3-3 demonstrates.

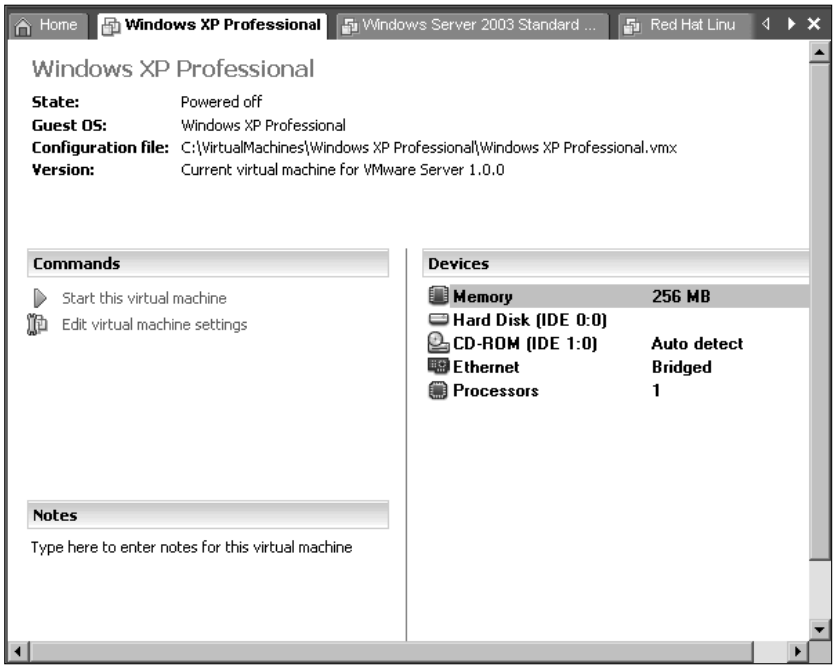

Figure 3-3

The main pane as shown in the preceding figure is comprised of five main areas:

- **Tabs** are how you switch between the multiple guests on your server. Each tab consists solely of a single virtual machine. You will notice there is a tab in the preceding figure for every virtual machine in my inventory. If the screen is too cluttered you can close all the tabs but the guests you are working on by pressing the X in the upper-right corner. If you need to regain access to a tab, simply return to your inventory pane and select the guest you are interested in again.
- **Title and Status** is where the current power status of your guest is located. It displays the current state, the configured guest OS, its configuration file and version. You will spend little time gazing at this section unless you need a quick reminder of where the guest is located or its state if you happen to be in the configuration screen while the machine is powered on. It is, in effect, just a quick overview.
- **Commands** is home to the mighty power button. It is just one of several places where you can switch your guests on. Commands is also where you will most likely go to open the configuration settings to adjust specific aspects of your guests like hard disks, networking, memory, and so on.
- **Devices** is a quick overview of your virtual hardware. It provides a look at your hardware configuration without entering the machine settings screen from the Commands area. It is interactive and context sensitive, which allows you to, for example, double-click on *Memory* and it will open just the memory settings page of your guest settings. This functionality saves time by allowing you to drill down on just the one specific setting you wish to alter.
- **Notes** provides a quick scratch pad to jot down items of interest for the guest. An example (and something that I do quite frequently) is to list the service pack level installed on the guest operating system.

As you can see, the five areas that make up the main pane allow you to control almost every aspect of your virtual machines. The most important function of the main pane, however, is the display of the guest after it has been powered on. Once you power the virtual machine the tab will become the graphical display of your machine.

If you take another look at the preceding figure you will notice I skipped a very important tab, the *Home* tab. Figure 3-4 shows the Home tab. This tab is most likely where you will choose to create and open your virtual machines as well as configure certain host settings for your server.

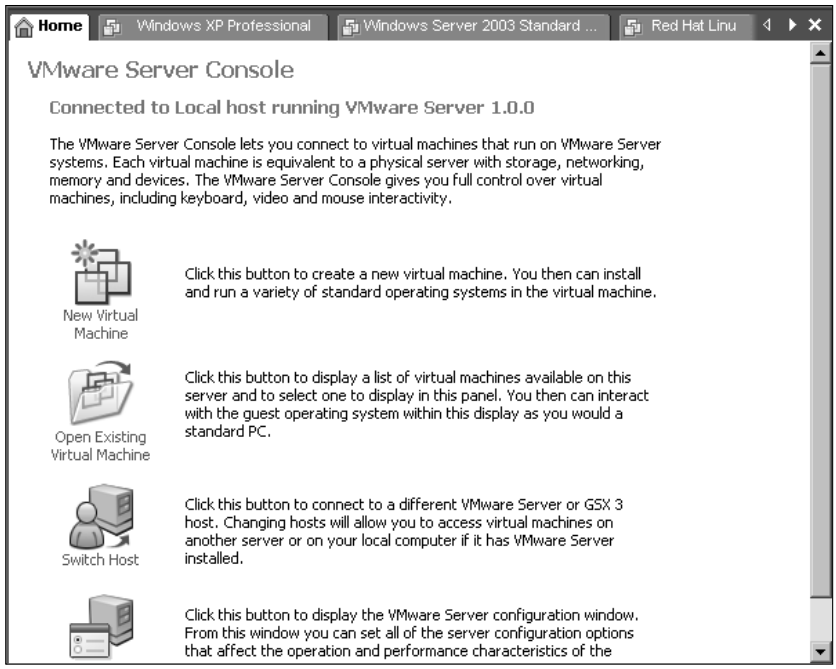
Home
Windows XP Professional
Windows Server 2003 Standard ...
Red Hat Linu
VMware Server Console
Connected to Local host running VMware Server 1.0.0
The VMware Server Console lets you connect to virtual machines that run on VMware Server systems. Each virtual machine is equivalent to a physical server with storage, networking, memory and devices. The VMware Server Console gives you full control over virtual machines, including keyboard, video and mouse interactivity.
New Virtual Machine
Click this button to create a new virtual machine. You then can install and run a variety of standard operating systems in the virtual machine.
Open Existing Virtual Machine
Click this button to display a list of virtual machines available on this server and to select one to display in this panel. You then can interact with the guest operating system within this display as you would a standard PC.
Switch Host
Click this button to connect to a different VMware Server or GSX 3 host. Changing hosts will allow you to access virtual machines on another server or on your local computer if it has VMware Server installed.
Click this button to display the VMware Server configuration window. From this window you can set all of the server configuration options that affect the operation and performance characteristics of the

Figure 3-4

- **New Virtual Machine** will present you with the *New Virtual Machine Wizard*. This wizard will guide you through the step-by-step process of creating a new virtual machine on your host machine. This will be become a repetitive task as you are creating your base images later in Chapter 4.
- **Open Existing Virtual Machine** provides another method in which you can select machines from your existing inventory or even browse for machines on your host that are not yet in the inventory. Once you select a machine using this method it will display in a tab just as it would coming straight from the Inventory pane as described previously.

If you browse for a virtual machine not yet in inventory from a remote client, keep in mind that the file paths shown are relative to the host and not your local workstation.

- **Switch Host** allows you to switch your console to another VMware Server installation elsewhere. This functionality finds use when utilizing remote clients connecting to hosts via your local network and there are multiple VMware Servers available. In the case of single server installations you will find little use for this feature.
- **Configure Host** allows you to access the settings panel for your host server. The various options available in host settings are covered in a later section.

The toolbar and menus, like many applications, provide access to various settings, the display of specific panes that have been closed, another way to start and stop your guests, and so on. All the items found on the toolbar have corresponding menu items just as the items in the main pane described previously. I am not going to go through each individual menu here. I will leave that for you to explore. I will, however, be pointing out specific menus throughout the rest of this chapter that pertain to host settings, hardware, and a few other pertinent things.

Host Settings and Console Preferences

Two types of settings are available in your server console: host settings and console preferences. Let's take a quick overview here and then move into more specific aspects in the sections that follow:

- **Host settings** are specific to your host server only. These include the memory pool available to virtual machines, the default location of your virtual machine files, and whether to use SSL to secure communications between the host and remote client. If the host is a Windows Server you will also have a choice for input priority.
- **Console preferences** pertain to your console environment and how it is set up. These are per machine so they affect only the remote client installed locally. Some of the items you have control over are update checks for your console, input grabbing and how the console reacts to mouse movements, hot keys, and finally display resizing for your guests.

Host Settings

The host settings available differ by the server platform used. A host server running Windows divides the settings into five categories: General, Memory, Priority, Devices, and Connection.

If your host is running on Linux you will see only three categories, General, Memory, and Connection. The Priority setting is not available on a Linux host. Because this difference is relatively minor, I will cover the host settings for a Windows-based server. If your server is Linux based, just ignore the priority settings information.

General

Figure 3-5 shows you the first settings category — not too exciting but still important.

You can access the following host settings window by selecting the menu item Host and choosing Settings, or by clicking the Host Settings button on the Home tab in the main pane.

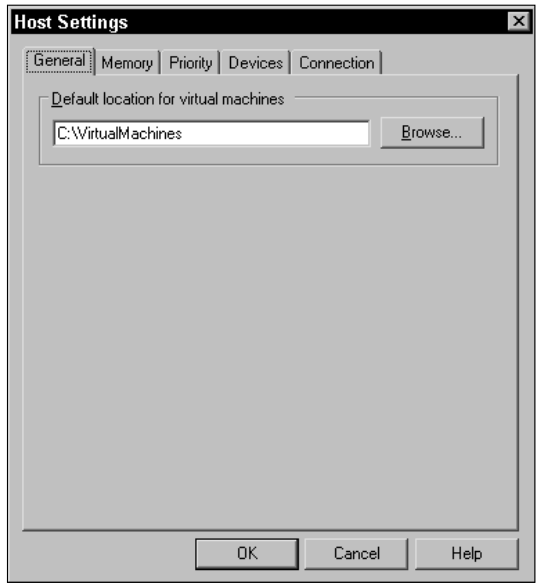

Figure 3-5

The *General* tab offers only one setting, the default location of your virtual machines. Because I am using a Windows-based host in this example, you'll notice the path of `C:\Virtual Machines` is used. If you would like to use a different path for the default location, type it in the textbox or click the Browse button.

If you use the Browse button and you are on a remote client, keep in mind that the directories and files shown are relative to the host server itself and not the local machine.

If you are on a Linux-based host, the default location, unless changed during install is `/var/lib/vmware/Virtual Machines`. If you change this location you will need to ensure the proper security settings are applied to the new directory. More on that

Memory

Next is the *Memory* tab. This group of settings, as shown in Figure 3-6, allows you to configure how much system memory to allocate for all running virtual machines as well as how to handle memory swapping for each machine.

I should point out right off that I only have 1GB of system memory installed on this machine. VMware Server automatically reserves a small amount of that for the host operating system. After all, you don't want your virtual machines to use all of the memory available. You can move the slider down, which will allow the host server more memory for its own processes, or you can leave it turned all the way up, which means your virtual machines can use all but, in this example, 100MB of your host's memory. For now I'm going to leave this setting as is because I don't plan on running a large number of guests on this machine.

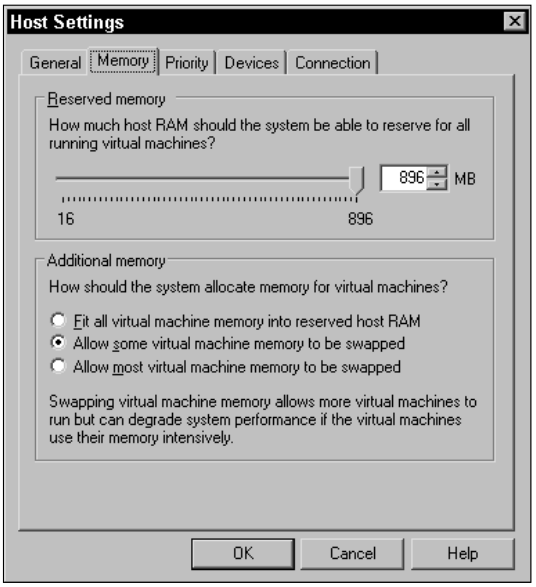
Host Settings
General
Memory
Priority
Devices
Connection
Reserved memory
How much host RAM should the system be able to reserve for all running virtual machines?
896
MB
16
896
Additional memory
How should the system allocate memory for virtual machines?
Fit all virtual machine memory into reserved host RAM
Allow some virtual machine memory to be swapped
Allow most virtual machine memory to be swapped
Swapping virtual machine memory allows more virtual machines to run but can degrade system performance if the virtual machines use their memory intensively.
OK
Cancel
Help

Figure 3-6

The default setting is to use all available memory on the host minus the amount VMware Server reserves.

The setting under *Additional memory* determines how the virtual machine's memory will be handled by the host server memory management system. Like most operating systems, physical memory gets swapped out to the hard disk to make room for more items in memory and to increase performance. This swapping action can have an impact on guest performance for the simple fact that it takes time to swap memory to and from disk. Honestly, in my experience the default setting to allow some of the memory to be swapped is acceptable. You can, however, adjust this if you are having issues with performance in your guests or are low on system memory.

Priority

Next is the *Priority* tab. Remember, this is available only on Windows-based hosts. If you run VMware Server on a Linux host you will not find this in the host settings window. Take a look at the tab in Figure 3-7.

A Windows-based host allows you to prioritize your processes in order to either escalate or deescalate their performance. By default most all processes on a Windows-based system are set to a Normal priority. This means that the process must contend with other processes set at a Normal priority for system resources. If you choose to escalate a process to High, for example, you essentially move it to the front of the line ahead of the Normal and lower priority processes. This allows the process to execute more quickly by having first dibs on the resources of the host.

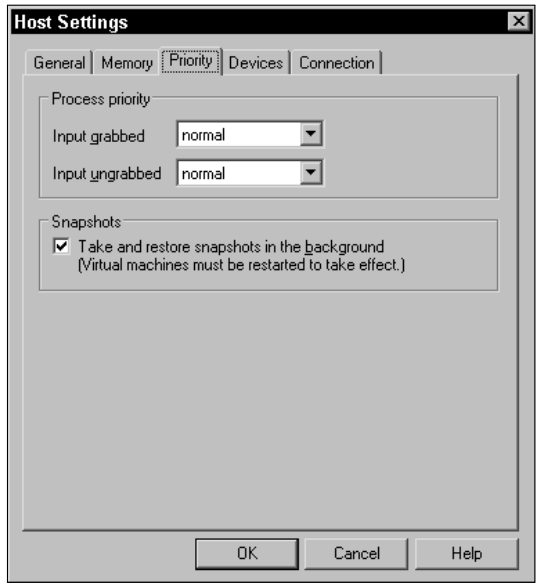

Figure 3-7

VMware Server allows you to set the process priority for your virtual machines based on two states: Input grabbed and Input ungrabbed. This simply refers to whether or not the virtual machine console has the input focus of either the mouse or keyboard. Click outside of the Virtual Machine console or press Ctrl+Alt, or the standard hot key you defined if other than the default, and the input is now ungrabbed and the priority set for the Input ungrabbed state takes hold. I recommend you use this setting with caution. Setting your virtual machines to a higher priority on the host system can cause them to starve the host machine of resources, and you can experience adverse effects. It does, however, have its applications, so I'll leave it to you to determine whether or not a specific virtual machine needs a higher priority on your host server.

One final thing about priority before I move on. The settings shown in Figure 3-7 are system-wide only for the user logged into the VMware Server Console. They will affect all virtual machines run under the security context of that logged-on user. If another user logs in to the console, the priority settings are once again back to their defaults. It is possible to set the virtual machine priority on each individual machine. Setting the priority locally, in the virtual machine properties itself, overrides the global setting at all times. If you have a virtual machine set to use high-normal (grabbed-ungrabbed), then a setting of normal-normal in the global setting will have no effect for that machine. Priority settings for machines that are set to run under the Local System account must be made within the virtual machine configuration itself. Global settings, because they are per user, will not apply to the Local System account.

Devices

The *Devices* tab is rather simple. If you remember back to Chapter 2 the installer asked you if you wanted to disable CD-ROM autorun for the host machine. I advised you to go ahead and disable it to avoid any funkiness when a CD-ROM is inserted into the host machine. This tab, shown in Figure 3-8, allows you to either set or reverse that setting. The following figure shows the state if you decided to let the installer disable autorun for you. If you want autorun back to its default state of on, simply uncheck the checkbox. It may require a reboot of the host machine for the change to take effect.

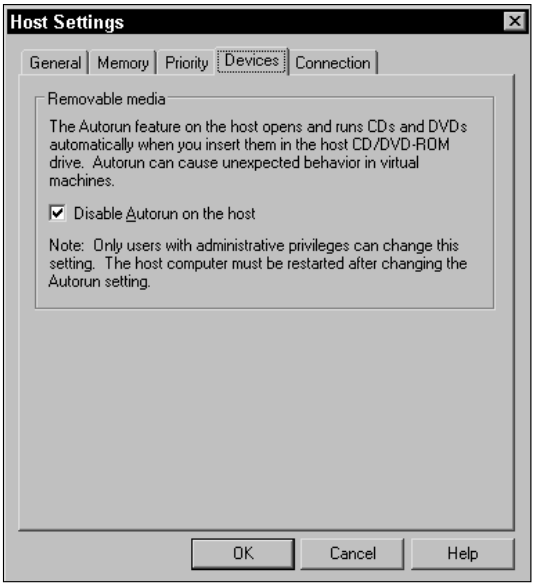
Host Settings
General
Memory
Priority
Devices
Connection
Removable media
The Autorun feature on the host opens and runs CDs and DVDs automatically when you insert them in the host CD/DVD-ROM drive. Autorun can cause unexpected behavior in virtual machines.
Disable Autorun on the host
Note: Only users with administrative privileges can change this setting. The host computer must be restarted after changing the Autorun setting.
OK
Cancel
Help

Figure 3-8

Connection

Finally, there is the *Connection* tab as shown in Figure 3-9. From here you can choose if you want the remote client's console connection to the server encrypted. This is handled by a self-signed SSL certificate generated at install. The default is to use SSL encryption and requires no other interaction on your part.

Those are the host settings in a nutshell. Really, there's not a whole lot to configure but it is important to be familiar with a few items, such as memory management. The next section moves onto the console preferences.

Console Preferences

Just as any other Windows or Linux application has preferences, the VMware Server Console is no different. The console preferences control just about everything you will experience while using the console to interface with your guest machines. Take a look at Figure 3-10. The sections that follow explain the preferences in more detail.

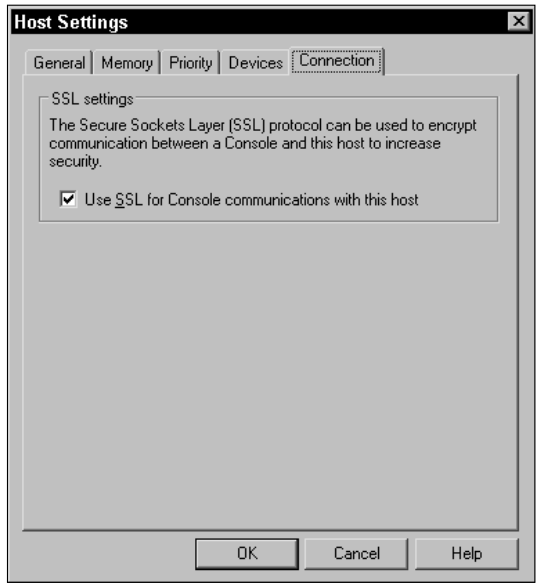

Figure 3-9

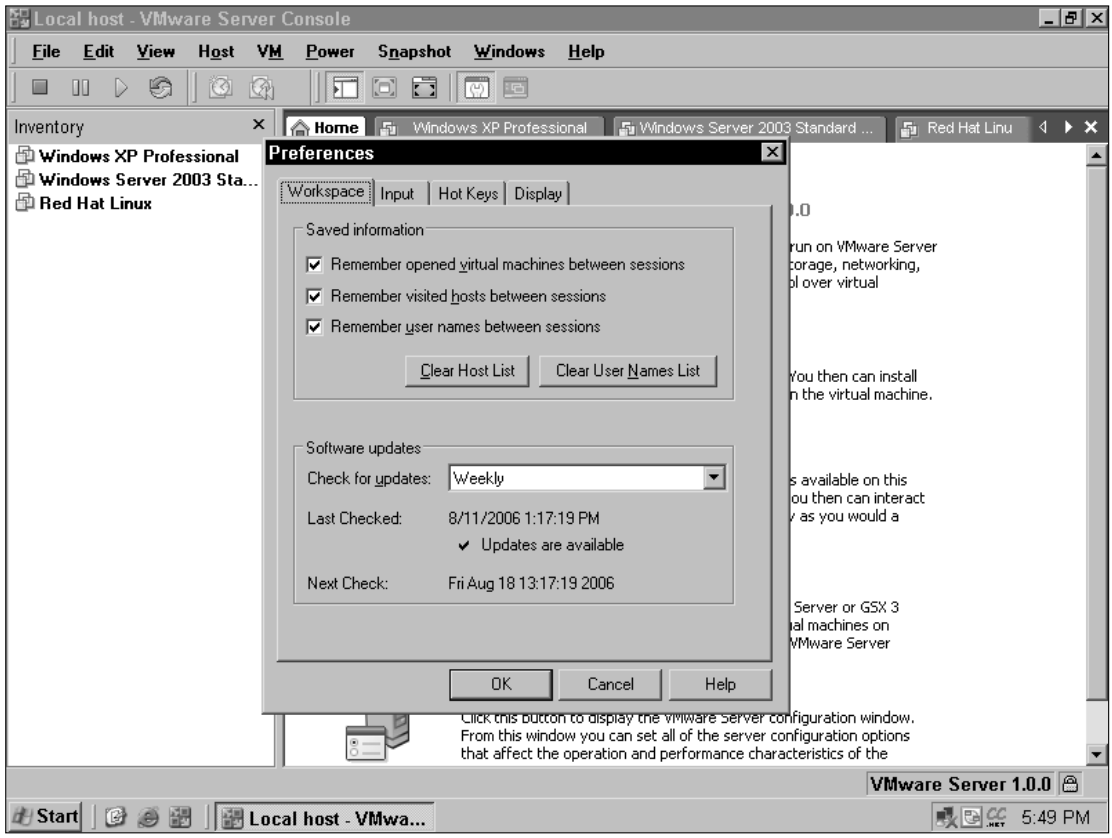

Figure 3-10

Workspace

The *Workspace* tab just offers you some simple behavior control over remembering certain information between sessions and checking for updates. I tend to leave the saved information items checked just to make my experience all the more convenient. However, you may, depending on your environment and security concerns, wish to disable the ability to remember host names and user names. You will see the effect of this at the host login screen presented when you first run the server console.

Software updates are just that, updates. The console will check at the interval you specify if there are any updates available to your software. Remember, however, that you are operating in a two-tier environment. Updating the console to the latest version should be done only in conjunction with a server update. The reason is simple: compatibility. Although an update to the VMware Server software may still work with a past version of the VMware Console installed on a remote client, you cannot always guarantee that they are completely compatible. It is also possible that a VMware Server update installs additional functionality that will not be present or even recognized by an out-of-date console. For this reason I recommend you always keep the server and remote clients at the same version level.

Input

The *Input* tab controls how your keyboard and mouse, cursor, and clipboard work between your local host and the console. Figure 3-11 shows the Input tab and its settings.

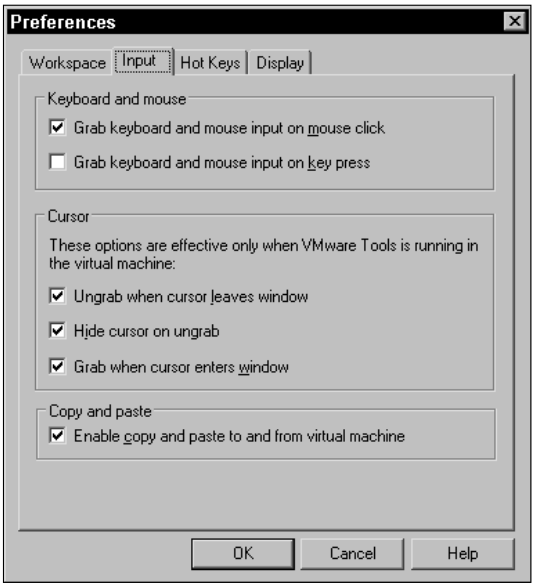

Figure 3-11

The first section on this tab deals with your keyboard and mouse. If you have never used the console to display a guest before, this will be a little hard to understand. Your mouse can have focus on your local desktop or the guest desktop but not both. Prior to the installation of the VMware Tools package you

will have to click inside the guest window in order to give focus to the virtual machine. Once the tools package is installed, this focus is passed automatically between the local desktop and the guest. The same goes for the keyboard. You wouldn't want to type on your local machine and have the key presses passed to the guest as well, so the keyboard also has to switch focus. The focus is passed as one unit. If you click inside the guest window and the mouse takes focus for the virtual machine, then so does the keyboard. The grab on mouse click selection does just that; when you click inside the guest window focus is given to the virtual machine. If you select grab on keyboard press, focus will shift whenever you press a key and the guest window is selected and in the foreground. The default is to grab on mouse click and will most likely suit your needs. I encourage you to try them both to see which you prefer.

> *VMware Tools is a guest package that should be installed on every virtual machine you create. Both Windows and Linux versions are available for use. VMware Tools provides drivers and a better overall experience while using your virtual machines by automatically handling input focus, drag and drop, and various other things. I cover the installation and function of the VMware Tools package more in Chapter 4.*

The next section of the tab determines the cursor behavior when the VMware Tools package is installed. The selections are pretty self-explanatory, so I won't dive into them. The three options determine whether the console will automatically shift focus to the guest when the mouse passes over the console window or if you have to click inside the console to change focus. The default selection is all three checked. This offers the best overall experience.

The copy and paste section wraps up the Input tab. With this checkbox selected, the server console will capture your local machine's clipboard input and allow you to paste it inside the guest machine. You can try this out by selecting some text on the local machine and pasting it inside the guest machine. You will find the ability to copy and paste between machines quite useful. The default is to allow copy and paste between the guest and local machine.

Drag and Drop in VMware Server

While I am on the subject of copy and paste, I want to talk briefly about drag and drop. In Windows it is quite common to drag an object, be it a file or something else, and drop it within another window or application. This holds true with the server console. If you have, for example, a text file on your local desktop and you want it in your guest machine you can simply drag the text file from your local desktop into the guest machine's console window. If Windows is running as the guest operating system as well, the console will transfer this file from your local machine to the guest's desktop.

Drag-and-drop functionality unfortunately is limited to Windows on Windows. What I mean by that is if your console is running on a Windows-based machine and your guest is also a Windows-based machine and VMware Tools is installed, drag and drop will work perfectly. It will not work, however, if any or all of the above operating systems are not Windows-based. You cannot, for example, drag a file from your Linux-based local machine and drop it into a Windows-based host. This is simply a limitation of the operating system and how it handles drag and drop. However, I find drag and drop extremely useful with Windows-based systems.

Hot Keys

Next, if you click the *Hot Keys* tab, you will see a window similar to Figure 3-12.

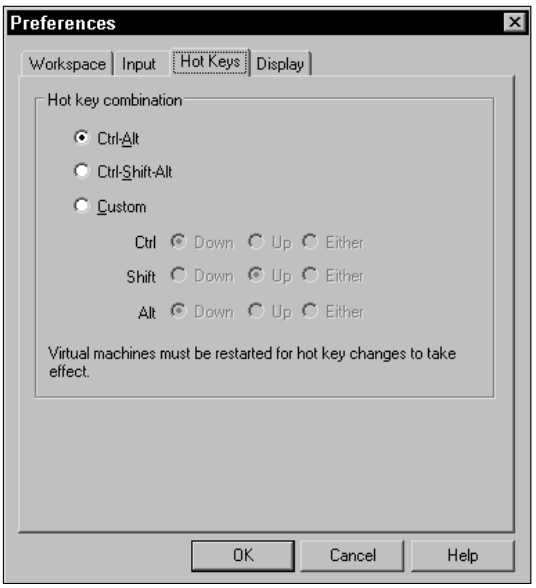

Figure 3-12

The Hot Keys tab controls what you want your key combination to be to do things like grab focus away from the guest, switch between virtual machines, and so on. Some of the ways you can use hot keys throughout the console include the following:

- Switching between virtual machines
- Un-grabbing input from the guest
- Leaving full screen mode
- Sending the Windows security keystroke to the guest (Ctrl+Alt+Del)
- Sending other combination keystrokes to the guest and not the host

Probably the first time you will notice the hot key settings is when you try and press Ctrl+Alt+Del to log in to a guest Windows operating system. This keystroke will send Ctrl+Alt+Del to your host instead of your guest. You will find this somewhat annoying until you get the hang of it. Because this is one of the more common combination keystrokes, VMware created a special stroke, Ctrl+Alt+Ins, to send the Windows security command to your guest. Now, you are given the ability to change this special hot key. If you choose to use the Ctrl+Shift+Alt, the preceding Windows security keystroke would be Ctrl+Shift+Alt+Ins instead.

Some find changing the default hot key to something other than Ctrl+Alt provides a simpler interface because you don't have to worry about sending things like the security keystroke to your host instead of your guest. Accordingly, you can customize this further by using a custom keystroke. For example, you

can choose Custom in this tab and have Ctrl and Alt selected for the down position and Shift for either. This means you can either use Ctrl+Alt or Ctrl+Alt+Shift as your keystroke. The list that follows describes the different choices for each key modifier in the custom setup:

- **Down** — The key must be depressed during the hot key sequence
- **Up** — The key must not be depressed during the hot key sequence
- **Either** — The key may either be depressed or not during the hot key sequence

If you select either for all the keys, then the Esc key is used to leave full screen mode.

I already covered how to send the Windows security sequence to the guest; however, there is one more that you will use quite often and that is changing to and from full screen mode. If your guest is displayed in a window within the console, you can press Ctrl+Alt+Enter while the guest has the focus to switch to full screen mode. The hot key sequence of just Ctrl+Alt will bring the guest back to windowed view.

I find the other key shortcuts are seldom used, but feel free to view the VMware Console help for additional key sequences that you can use to manipulate your guest machines.

Display

To wrap up the console preferences take a look at the *Display* tab, as shown in Figure 3-13.

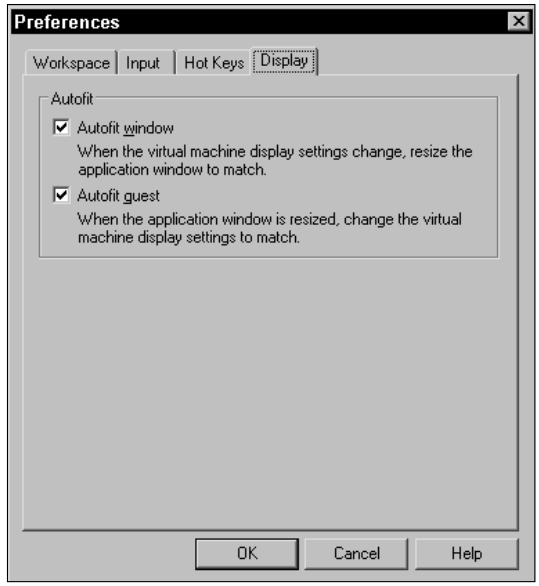

Figure 3-13

These settings in combination with the installation of VMware Tools on the guest machine will control how your guest screen resizes when switched to either the quick switch or full screen view. As you will see once you power on your first virtual machine, the virtual machine defaults to being displayed within the console window. You can switch this view two ways:

- The **quick switch** mode is where the full screen is used but you still have access to your virtual machine tabs, the toolbar, and the console menu bar, which is hidden from view at the top.
- The **full screen** view is where the guest machine takes the entire screen just as any locally installed operating system would. Sometimes in full screen mode you will forget you are even in a virtual machine.

The different views can be accessed from either the View menu or icons on your toolbar.

The *Autofit* section of this Display tab offers two choices: Autofit window or Autofit guest. As you can tell from the descriptions provided in the settings window, the default is to alter the console window to match a resolution change in the guest. This can sometimes result in guest console windows bigger than your actual screen if the guest is at a higher resolution than your local machine. The second choice, Autofit guest, will resize the guest via the installed VMware Tools to match the size of the console window. You will find it is better to leave the default checked, Autofit window, and just set a resolution on your guest machine that fits within your local machine's resolution. Typically you will work with your guests in a full screen or quick switch mode anyway.

Virtual Machine Settings

For virtual machines to be of much use you need virtual hardware. The basics are set up during the New Virtual Machine Wizard that you run through as you create a new virtual machine. I want to take a few moments to run through what is available under the *Edit virtual machine settings* option. If you refer back to Figure 3-3 I am going to select Edit virtual machine settings under the Command section. You can also select VM➪Settings from the menu bar with the virtual machine tab selected. Figure 3-14 shows the Virtual Machine Settings window.

I cover how to create a virtual machine and the New Virtual Machine Creation Wizard in Chapter 4.

Each tab on the Virtual Machine Settings window is made of two panes, the category selection on the left and the matching settings pane on the right. In the preceding figure you will notice I have the virtual machine's memory selected. In the right-hand pane you can see that you are able to adjust the available memory for your guest. Here I am allocating 256MB of RAM to the virtual machine. VMware Server will always default this value to the minimum recommend amount from the operating system manufacturer. In addition to memory the left-hand pane shows various other pieces of hardware that the currently selected virtual machine has available to it. Take a few minutes to click through the various virtual hardware items that were set up by default to get a feel for the settings available for each. Once you do that I want to take a few minutes to go over with you just what hardware options are available for your virtual machine's use and how they present themselves to the guest operating system.

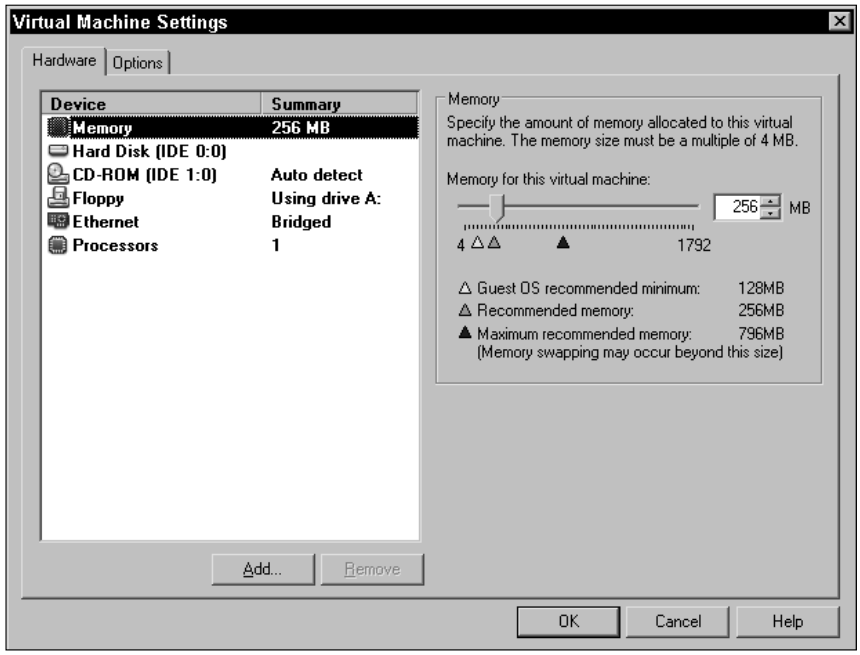

Figure 3-14

Virtual Hardware Specifications

Each virtual machine created with VMware Server has the following possibilities for virtual hardware.

The following information was taken from the VMware Server Virtual Machine guide. For the most up-to-date list of exposed virtual hardware please reference that document.

- **Virtual Processor**
  - Intel Pentium II or later, or AMD Athlon or later. The processor exposed to the virtual machine depends on the host's processor.
  - On machines with multiple processors or dual core processors the ability to expose more than one processor is supported.

The list of virtual processors does not indicate that your virtual machines will see only a Pentium II or Athlon. Rather, the virtualization layer virtualizes the host processor and reports its type and specifications to the guest machine. The guest will see the actual manufacturer and model of the host processor.

- **Virtual Chip Set**
  - Intel 440BX based motherboard with NS338 SIO chip and 82093AA IOAPIC
- **Virtual BIOS**
  - PhoenixBIOS 4.0 Release 6 with VESA BIOS
  - DMI/SMBIOS compliant for system management agent support
- **Virtual Memory**
  - Each virtual machine can have up to 3.6GB of system memory dependent on the host system's specifications and limitation of the guest operating system.
- **Virtual Graphics**
  - Each guest machine is offered a VGA- or SVGA-capable video card.
- **Virtual IDE Drives**
  - CD-ROM drives can be an ISO image or a physical drive on either the host or local system.
  - Hard disks can be virtual, consisting of a file on the hard disk of the host, or an actual physical hard disk partition.
  - IDE disks have a maximum size of 128GB each. It is possible to have more than one virtual IDE disk per virtual machine, each with a maximum of 128GB.
  - A total of four IDE devices may be configured for each guest. This can be any combination of hard disk, CD-ROM, or DVD-ROM drive.
- **Virtual SCSI Devices**
  - Offers either a BusLogic BT-958 compatible host bus adapter or an LSI Logic Ultra 160 LSI53C10xx SCSI controller.
  - Generic support for scanners, CD-ROM, DVD-ROM, tape drives, and other SCSI devices with the use of guest operating system drivers.
  - A total of no more than 60 SCSI devices on up to four virtual SCSI controllers.
  - SCSI disks have a maximum size of 256GB each. It is possible to have more than one virtual SCSI disk per virtual machine, each with a maximum of 256GB.
  - Hard disks can be virtual, consisting of a file on the hard disk of the host, or an actual physical hard disk partition.
- **Virtual PCI Slots**
  - A total of six virtual PCI slots are divided among the virtual SCSI controllers, virtual Ethernet cards, virtual display adapter, and virtual sound adapter. You cannot configure a virtual machine to use more virtual hardware devices than virtual PCI slots.
- **Virtual Floppy Drives**
  - You may have up to two regular 1.44MB floppy drives.
  - The floppy drives can be a physical drive or a floppy image file.

- **Virtual Serial (COM) Ports**
  - You may have up to four standard serial ports.
- **Virtual Parallel (LPT) Ports**
  - You may have up to three bidirectional parallel ports.
- **Virtual USB Ports**
  - A two port USB 1.1 UHCI compatible controller is provided. USB 2.0 is not supported for virtual machines at this time.
- **Virtual Keyboard**
  - A standard 104-key keyboard is emulated.
- **Virtual Mouse**
  - PS/2 or serial table support.
- **Virtual Ethernet Card**
  - You may have up to four virtual Ethernet cards per guest.
  - Offers the AMD PCnet-PCI II chipset.
  - Wireless networking is supported through the use of bridged and NAT networking with the host machine.
  - Supports PXE ROM version 2.0.
- **Virtual Sound Adapter**
  - Offers a Creative Labs Sound Blaster AudioPCI chipset. (MIDI, game controllers, and joysticks are not supported.)

As you can see from the preceding list the available hardware for your use is quite lengthy. I talk in more detail in Chapter 4 about how to set up and configure specific pieces of hardware for use in your first virtual machine.

Now that you understand something about the hardware possibilities for your virtual machine, I want to move on to the options for your virtual machine available in the Virtual Machine Settings Options tab.

Virtual Machine Options

The Options tab contains the nuts and bolts of how your virtual machine will behave and operate. It controls how your machine will start up and shut down, the security of who can use your guest, the name and type of operating system installed, and many more options. The next sections take a look at this tab so you can see what you have.

General

Figure 3-15 starts out in the General section.

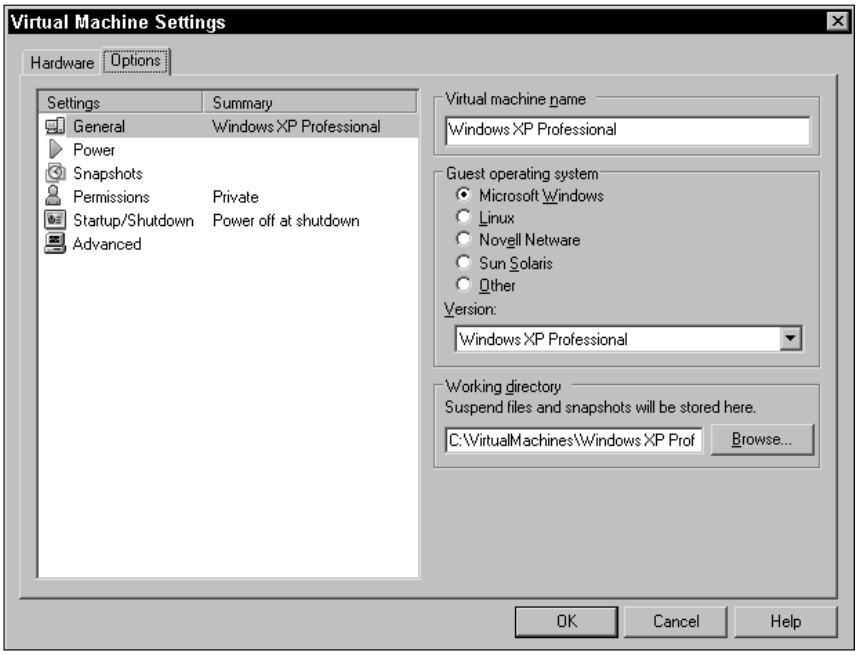

Figure 3-15

This section covers some basic settings that (except for the name) you most likely will not change after you initially create the machine. The virtual machine's name and guest operating system along with the version really need no explanation. Simply name the virtual machine what you like and set the appropriate choices for what you will be installing on the guest machine. The working directory should automatically be pointing to the location in which this specific virtual machine is stored on the host computer. If for some reason you move the location of your virtual machine after you create it, you may need to update this path.

Remember that the path is relative to your host server and not the local machine.

Power

Next, take a look at the *Power* section, as shown in Figure 3-16.

This section covers how the machine reacts to powering it on and off. The power options are self-explanatory as are the power controls that correspond to their respective toolbar icons. The Run VMware Tools scripts options, however, require a brief explanation. As you will see in later chapters, you can write scripts in Perl, C, or any language that supports a COM interface in Windows. These scripts can be used to perform various functions when a virtual machine is powered on, off, or suspended. These will come into play later as I discuss integrating guests into the software development and testing environment. For now remember that if you attempt to run a power script these options need to be enabled in order for the script to run properly.

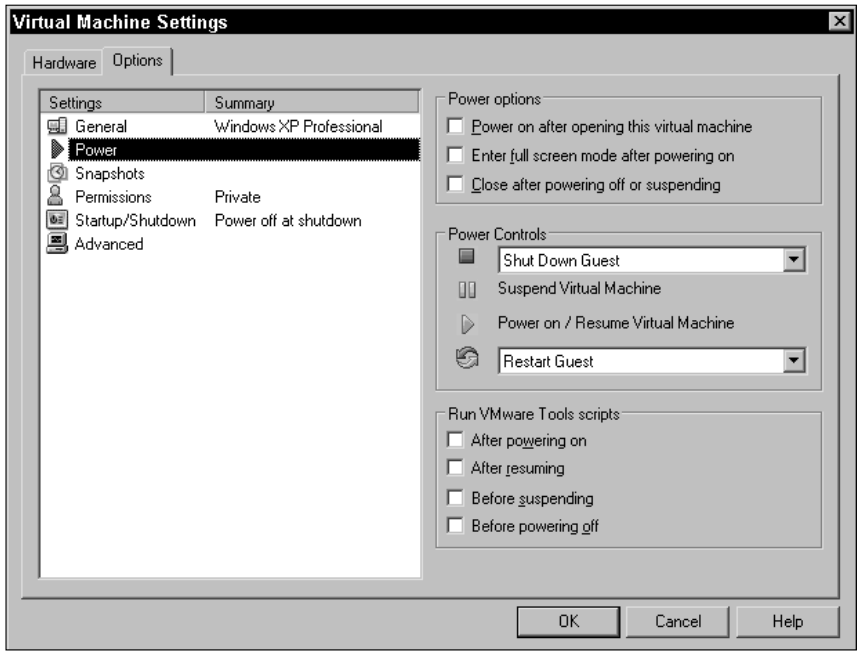

Figure 3-16

Snapshots

Next is *Snapshots* as shown in Figure 3-17. In Chapter 4 when you create your first virtual machine, the use of snapshots will be explained in greater detail. However, as a brief introduction, snapshots are used to freeze (so to speak) a virtual disk file and start tracking changes in a new disk file. It gives you the ability to rollback a virtual machine to a point in time the snapshot was taken. This will come in very handy later when you start installing applications to test on your virtual machines so that when you are finished, you can roll them back to a preinstallation point.

As Figure 3-17 shows, you have the ability to disable snapshots for a virtual machine or lock the machine into the current snapshot. You are only able to disable snapshots for a virtual machine that does not contain any previous snapshots.

Also notice at the bottom of Figure 3-17 that it asks what behavior to exhibit when powering off the virtual machine. You have three options:

- **Just power off** — This simply turns off the virtual machine making no changes to its state or snapshot.
- **Revert to snapshot** — This applies only if there is a snapshot to revert to. When powering off the virtual machine, it will automatically revert to the previously saved snapshot.
- **Ask me** — This option asks what you want to do out of the previous two options during shutdown. It allows some flexibility when testing or determining just how the machine behaves.

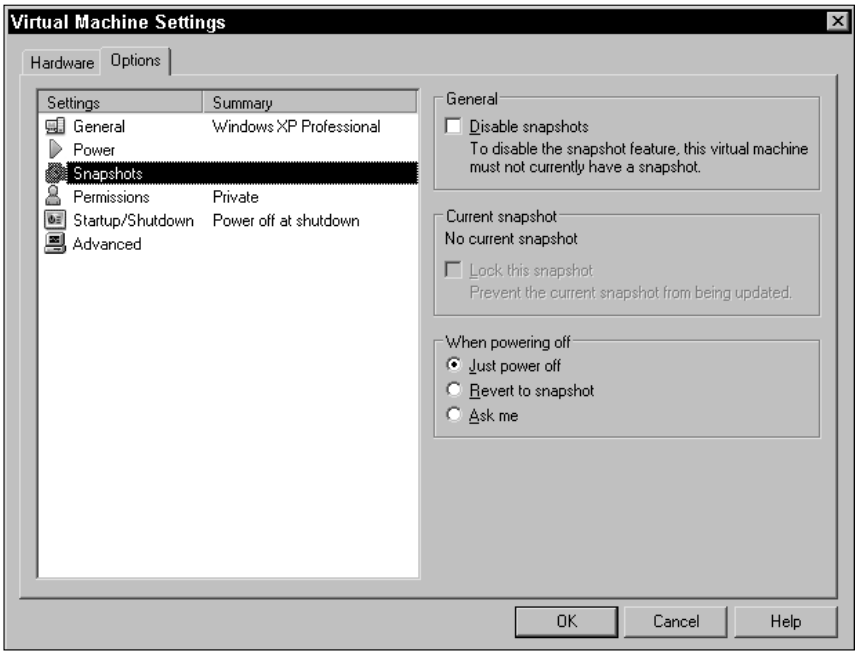

Figure 3-17

Permissions

Permissions in the settings menu offer you the ability to either allow multiple users to access a virtual machine or make a machine private and available only to the current logged-in user. Figure 3-18 shows the options available. Although the permissions setting seems like a one-stop shop for protecting your virtual machines from unauthorized use, it is in essence controlled by file security settings on the host server and not so much this setting. I get into this in more detail in the "Virtual Machine Security" section later in the chapter.

Startup/Shutdown

Startup/Shutdown allows you to control both what user the machine is run under on the server and the behavior of the virtual machine when the host is powered on. You can set your virtual machines to power on when the host is started up and shut down when the host is set to shut down. There is a delay built in so the host system has time to get fully going before loading down with a virtual machine startup. Figure 3-19 shows this setting.

You will notice that the Startup and Shutdown options are currently grayed out. This is because you need to set the virtual machine to run either as the local system account or a specific user. Once this setting is made, the Startup/Shutdown options become available, but they require a slight bit of host server configuration in order to adjust the built-in startup delays. I cover that shortly.

> *You can only change what account the virtual machine runs under if you are logged in to the console as a user with administrative rights on the host server.*

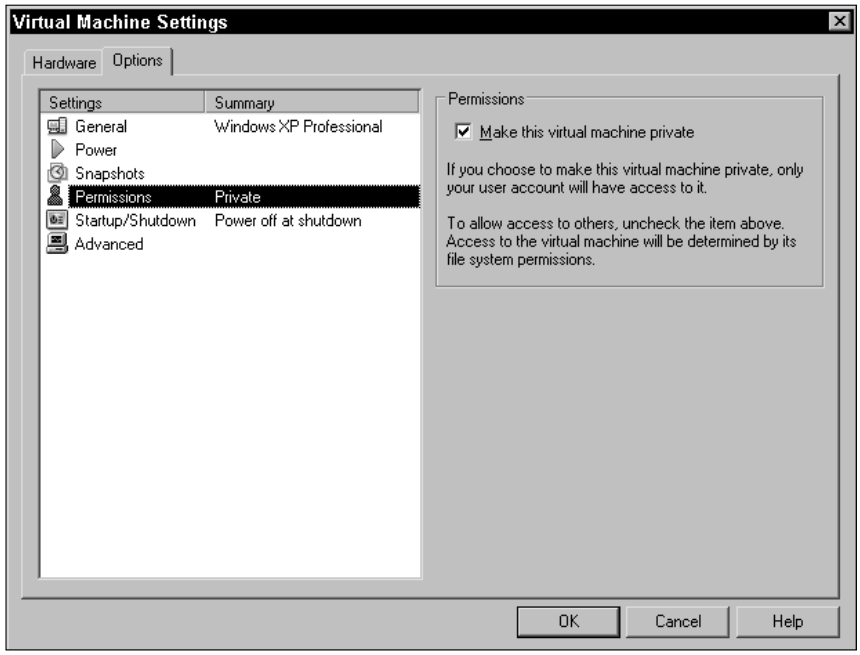

Figure 3-18

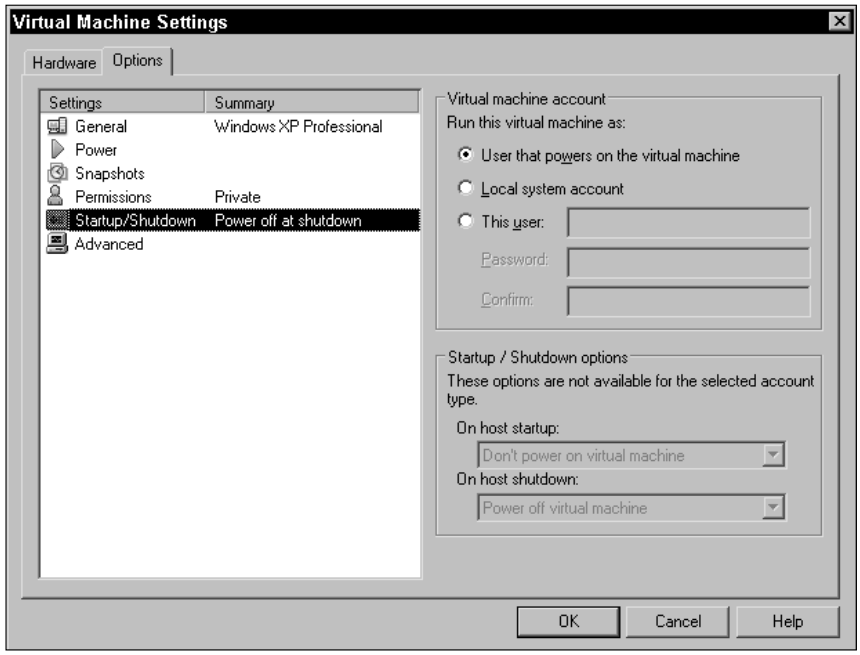

Figure 3-19

Without diving too much into what is covered in the "Virtual Machine Security" section later in the chapter, I should nevertheless talk briefly about the user context in which a machine is run. User rights to virtual machines fall into four basic categories:

- Browsing
- Interacting
- Configuring
- Administering

These four categories are controlled by file-level permissions on the host server. If you set the virtual machine to power on under the user context that starts it, some basic conditions have to be met. First, the user must have read and write permissions to the virtual machine files and, second, have the permissions in order to interact with physical devices on the server like USB controllers and SCSI devices. Again, more is covered on security later in the chapter, but for now I want to continue with the Startup and Shutdown configuration requirements for the host to get the machines to start up automatically.

Interestingly enough, the following startup delay settings must be made utilizing the VMware Server Management Console, the web-based console that you may or may not have installed. If you did not install the web management console back in Chapter 2, you will need to complete that step before proceeding. I have the console installed on Windows host `vmserver-01`, so take a look at what you need to set up. Figure 3-20 shows the web management interface. Browse to the URL of your host server's web interface and log in as the administrator. Remember, the interface is on a nonstandard port, port 8222.

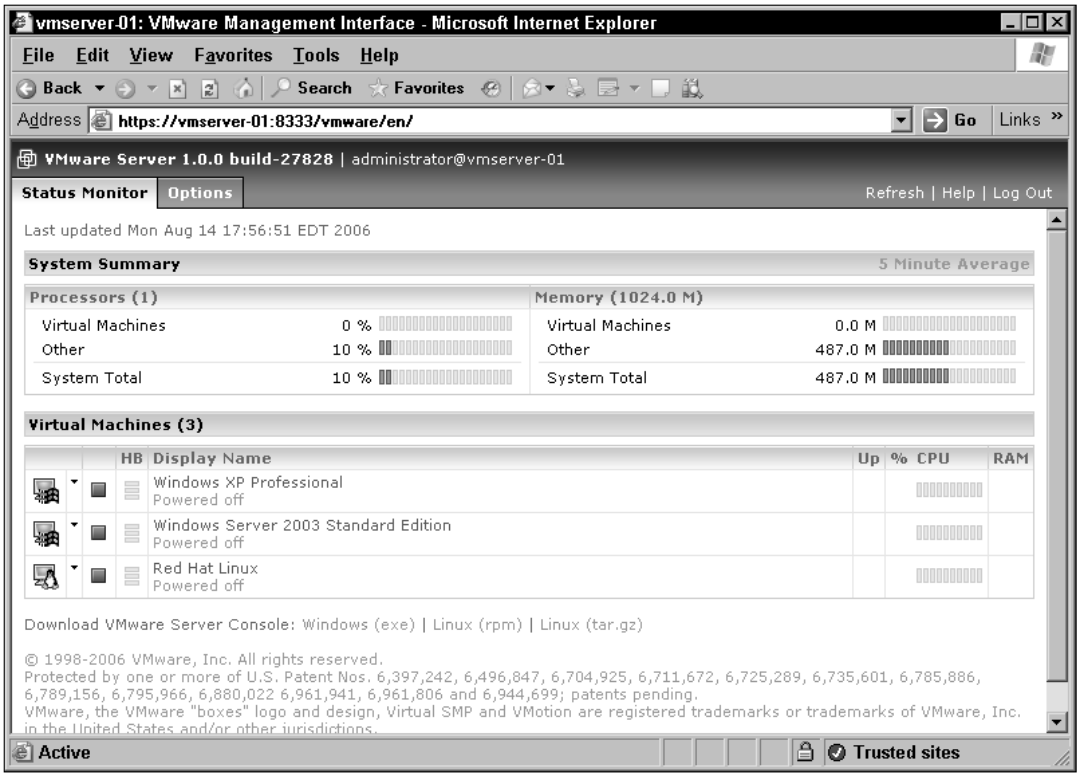
vmserver-01: VMware Management Interface - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites
Address https://vmserver-01:8333/vmware/en/
Go Links
VMware Server 1.0.0 build-27828 | administrator@vmserver-01
Status Monitor Options
Refresh | Help | Log Out
Last updated Mon Aug 14 17:56:51 EDT 2006
System Summary
5 Minute Average
Processors (1)
Virtual Machines 0 %
Other 10 %
System Total 10 %
Memory (1024.0 M)
Virtual Machines 0.0 M
Other 487.0 M
System Total 487.0 M
Virtual Machines (3)
HB Display Name Up % CPU RAM
Windows XP Professional
Powered off
Windows Server 2003 Standard Edition
Powered off
Red Hat Linux
Powered off
Download VMware Server Console: Windows (exe) | Linux (rpm) | Linux (tar.gz)
© 1998-2006 VMware, Inc. All rights reserved.
Protected by one or more of U.S. Patent Nos. 6,397,242, 6,496,847, 6,704,925, 6,711,672, 6,725,289, 6,735,601, 6,785,886, 6,789,156, 6,795,966, 6,880,022 6,961,941, 6,961,806 and 6,944,699; patents pending.
VMware, the VMware "boxes" logo and design, Virtual SMP and VMotion are registered trademarks or trademarks of VMware, Inc. in the United States and/or other jurisdictions.
Active
Trusted sites

Figure 3-20

> *If you are on a Windows Server 2003 host and browsing locally, you may need to add the local host to the trusted sites list in Internet Explorer. You can do so by opening Internet Options, choosing Security, and clicking the Trusted Sites tab. Enter* `https://<your server name>` *in the field and press Enter.*

I haven't talked much about the web management interface other than installing it, but you can do almost everything in this interface that you can in the remote console. As you can see, the inventory of virtual machines is listed along with their power status. For now I'm interested in the *Options* tab on top; click it to enter the options settings page as shown in Figure 3-21.

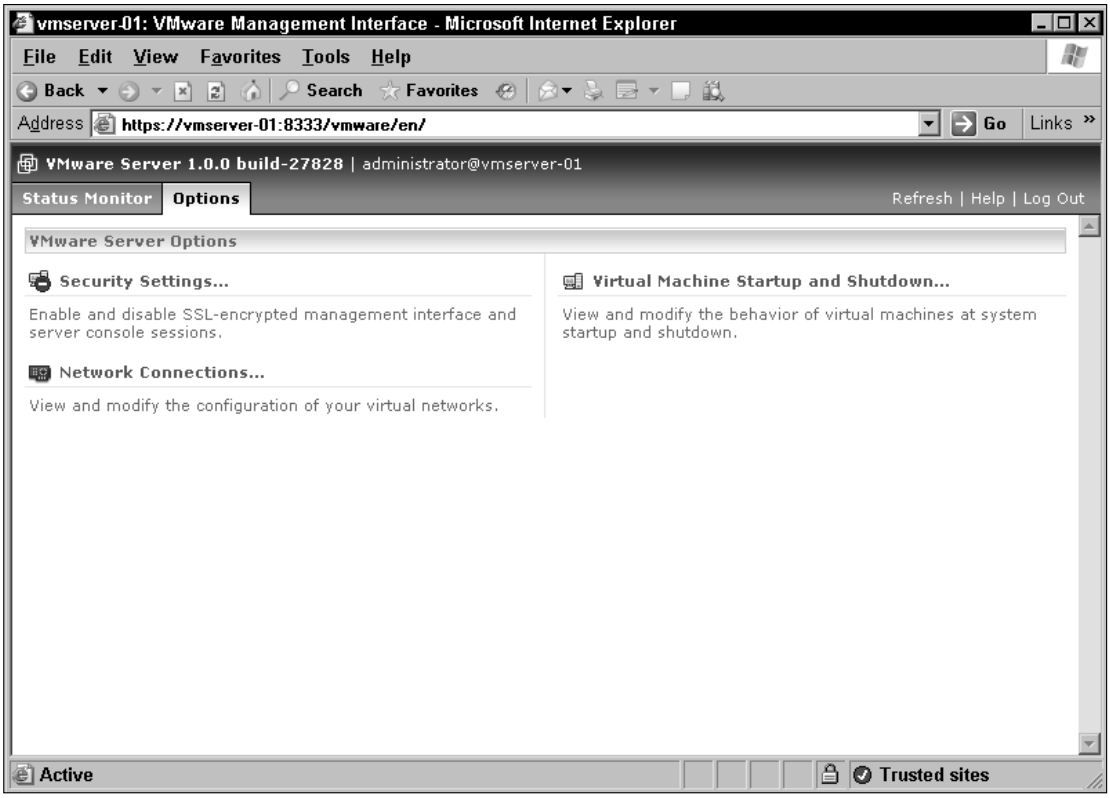
vmserver-01: VMware Management Interface - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites
Address https://vmserver-01:8333/vmware/en/
Go Links
VMware Server 1.0.0 build-27828 | administrator@vmserver-01
Status Monitor Options
Refresh | Help | Log Out
VMware Server Options
Security Settings...
Enable and disable SSL-encrypted management interface and server console sessions.
Network Connections...
View and modify the configuration of your virtual networks.
Virtual Machine Startup and Shutdown...
View and modify the behavior of virtual machines at system startup and shutdown.
Active
Trusted sites

Figure 3-21

The option you are interested in here is *Virtual Machine Startup and Shutdown*. Click it to view the page shown in Figure 3-22.

This is where you will set up your host to allow automatic startup and shutdown of your guests when the host server starts up or shuts down. The first time you enter this screen you may not have any settings visible under Properties. Click Edit under Properties to view the page in Figure 3-23, where you will configure the host.

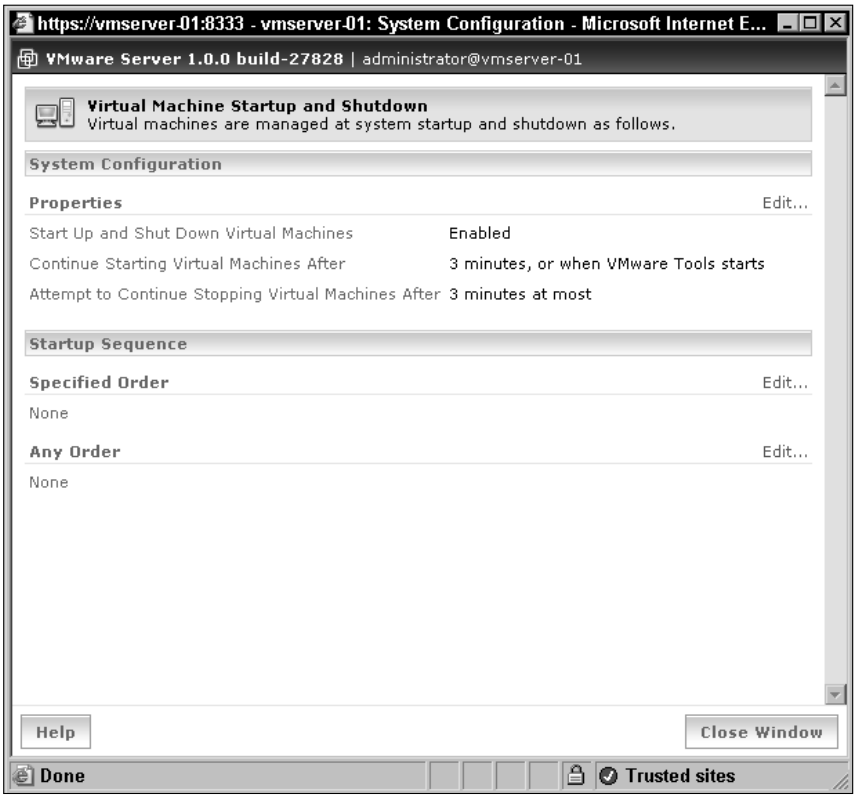

Figure 3-22

The default settings are shown in the preceding figure; however, you can modify the time delay for guest startup here. The important thing is to enable this by placing a checkmark in *Start Up and Shut Down Virtual Machines*. The time delay is up to you, but I prefer a longer startup delay, usually 4 or 5 minutes. This will vary on your host machine and its load. This is important because without a delay your virtual machines set to start automatically will all start up simultaneously, which can quickly bog down your host server. It may take you a couple of tries to find the right delay so everything starts up smoothly. Once you are finished, click OK and close the web management interface.

Advanced

Moving back to the Virtual Machine Settings window in the console, you are left with the *Advanced* section. Click it now and refer to Figure 3-24.

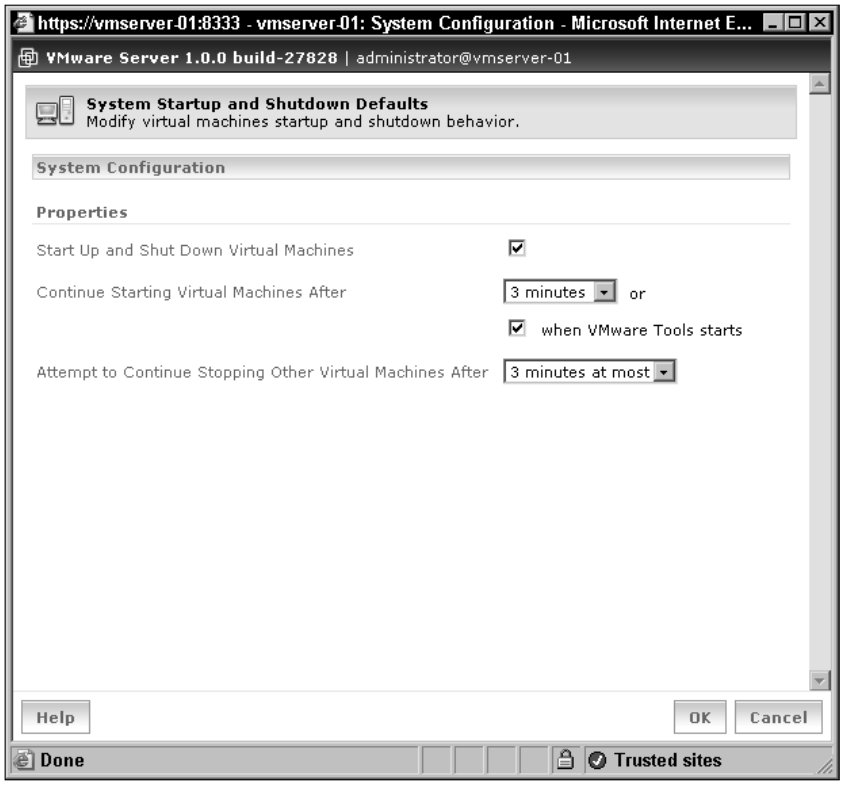

Figure 3-23

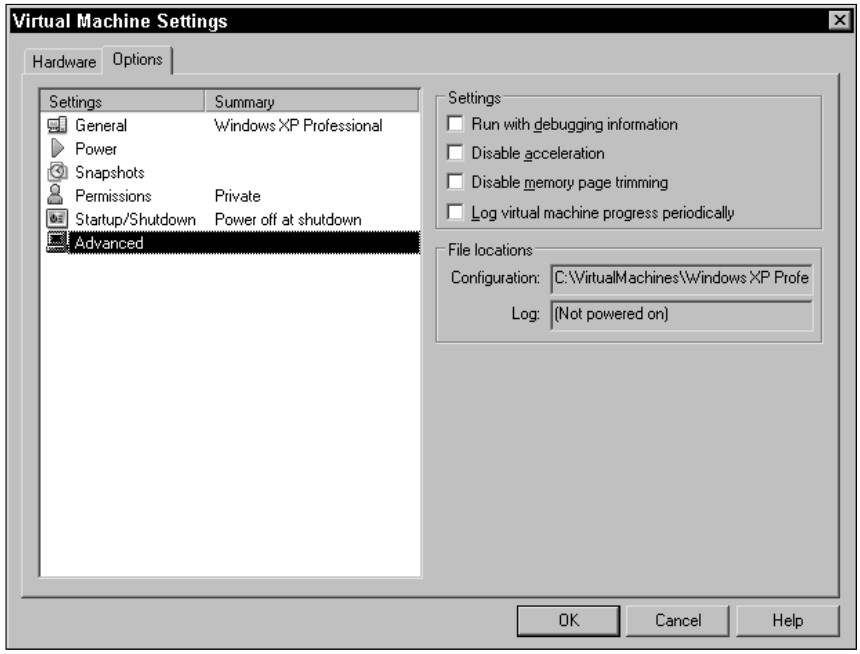

Figure 3-24

This offers you some advanced settings for your virtual machine's environment. For the most part only advanced users and special occasions will warrant the use of these settings; however, here's a brief list of their uses:

- **Run with debugging information** — This option, when selected, runs the virtual machine in a debugging mode. Virtual machine performance takes a huge hit in this mode; however, it does provide various debugging log files to be generated. For the most part the debugging log files are of little use to anyone but VMware Support.
- **Disable acceleration** — This one is a bit deceptive. The most common use for this is to check it if you have software that refuses to run or gives errors such as "cannot run under a debugger" in the guest machine. This option like the preceding one causes virtual machine performance to take a big hit.
- **Disable memory page trimming** — The VMware Server process itself continuously monitors the host and guest memory consumption. The process of trimming is where the server figures out how much memory the guest machine is utilizing and frees up some of the excess memory not in use to give back to the host server. This causes a slight performance hit to the guest when it attempts to access the freed memory and has to reallocate. The hit is, however, very small. Some state that by disabling memory page trimming they see a boost in virtual machine performance. It is up to you. The VMware stand on it is simply that if you have your host and guest memory planned out well, you can safely disable trimming. If you ensure that your host machine has an adequate amount of physical memory for all running virtual machines, including for its own overhead, then you should be okay. I never disable trimming myself, just to be safe.
- **Log virtual machine progress periodically** — The documentation is vague on this option. In my own experience, enabling it has caused logs to be generated at predetermined intervals that reflect the state of various virtual machine operations. The logging is not as detailed as the debug logs provided from the first option. Virtual machine performance takes a hit, though not as considerable a hit as full debugging. This option is best left to be used under the direction of VMware Support.

Running with debugging or periodic logging enabled will slow your machines down considerably. I recommend you use the settings under Advanced only to assist with support issues at the request of VMware Support.

Virtual Networking

Virtual networking, like networking in general, is how your virtual machines communicate with your network or the outside world.

> *This section presumes you have at least a basic knowledge of networking and the layout of your corporate network environment.*

For the most part you will typically be using only one kind of networking, bridged. You may, however, find it necessary to isolate your virtual machines from the rest of your network. This is where the network configuration comes in. Open the VMware Console and choose Host➪Virtual Network Settings. This opens the Virtual Network Settings window, as shown in Figure 3-25.

If your VMware Server is on a Linux host, this option is not available from the Host menu. Virtual Networking on a Linux host must be configured from the console using the `vmware-config.pl` *script.*

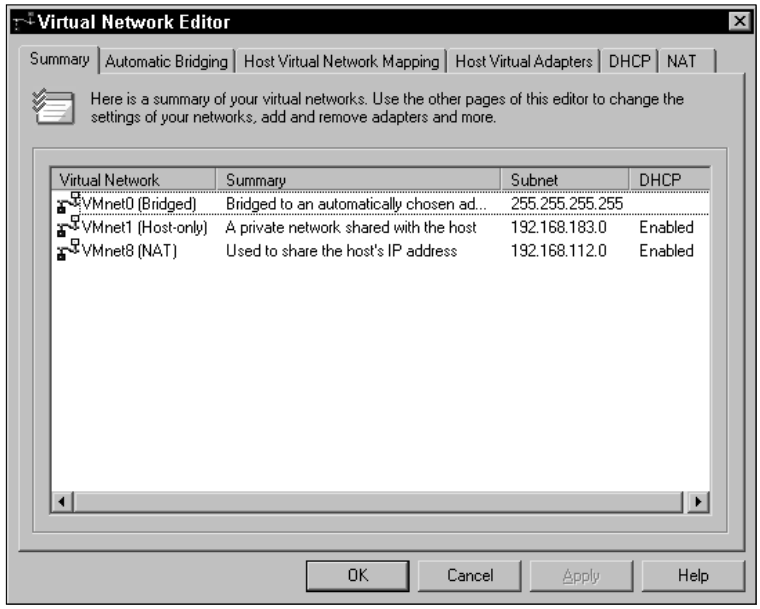

Figure 3-25

From the *Summary* tab shown in the preceding figure, you can see a list of the virtual network settings that were made for you automatically while installing the software. Over the next few pages I'm going to cover each tab and its function; however, configuring your virtual network and what type of networking setup you use will be left for you to consider based on your needs and current physical configuration.

As you can see, in the example I already have three virtual networks set up:

- **VMnet0** — A bridged virtual network to the host machine's physical network adapter.
- **VMnet1** — A host-only network where only the guest and host can communicate. This network is completely isolated from everything but the host.
- **VMnet8** — A Network Address Translation (NAT) network set up to share the host server's IP address on its physical network adapter.

The IP addresses shown in Figure 3-25 will be different on each installation; however, the text here will use the IP addresses for this particular install. Insert your IP addresses where appropriate as you follow along in the text.

Automatic Bridging

Now that you know what is already set up for you, you can move on to the *Automatic Bridging* tab as shown in Figure 3-26.

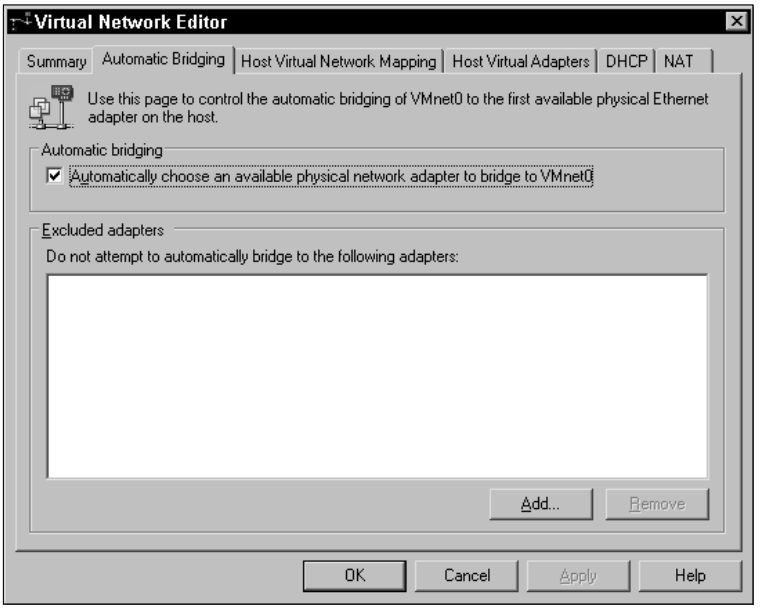

Figure 3-26

VMnet0 is always set up as your bridging adapter. Any virtual machine that is set up to utilize the VMnet0 interface will be bridged to your host server's physical interface. As I discussed in Chapter 2, this requires your network to have a DHCP server to issue the virtual machine an IP address or you to assign a static IP address to the guest once you install the operating system. From here you can have VMware automatically bridge VMnet0 to the first available network adapter on your host and choose to exclude certain adapters from that pool. You will see once you get to the *Host Virtual Network Mapping* tab how you can assign VMnet0 to a specific adapter on your server and not just allow it to remain automatic to the first available adapter. You can have odd results if you have more than one functioning network adapter on your host and you do not specify which adapter to use.

> *When you are utilizing any bridging virtual adapter for your virtual machines, it should be noted that they appear and act just as any other physical workstation on your network.*

Host Virtual Network Mapping

Next is the *Host Virtual Network Mapping* tab, as you can see in Figure 3-27.

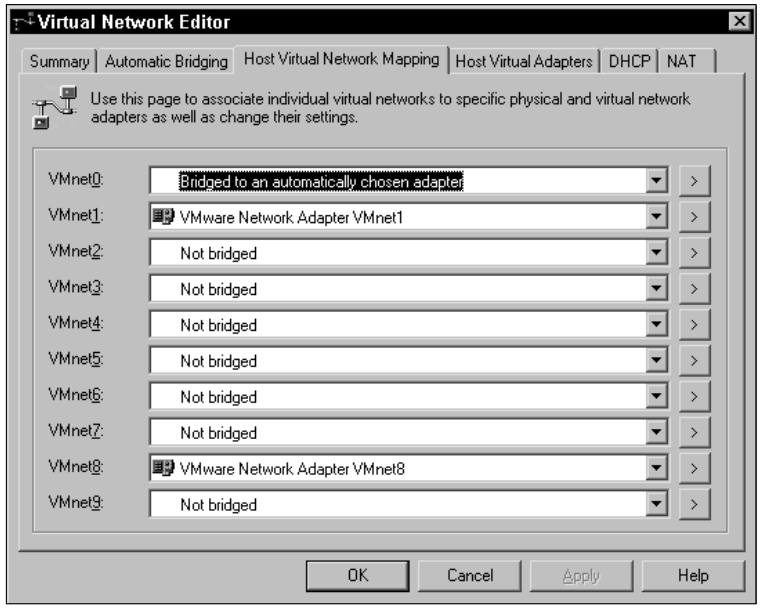

Figure 3-27

This is where you assign individual virtual networks to specific virtual network nodes from VMnet0 to VMnet9. Like I mentioned in the preceding section, you should first take a look at VMnet0. Right now you will notice it is set to bridge to an automatically chosen adapter. If you have only one physical adapter on your host server, this setting is fine; however, if you have two or more physical adapters on your host server, you may want to specify to which physical adapter VMnet0 is connected. Click the drop-down arrow and choose from the list of physical adapters; select the one you want associated with VMnet0. This becomes especially useful if you have multiple physical adapters on separate subnets. From here you can choose which subnet to bridge VMnet0 with and create another bridged connection with the other subnet if you so choose.

Now, as you may remember from the Summary tab in Figure 3-25, VMnet1 is connected to a host-only virtual network. You can configure the settings of that host-only network now. When you installed VMware Server, it searched for and selected a free subnet to place the host-only network on. In the case of the example server, vmserver-01, that network was 192.168.201.0. Here you can modify that subnet to something else. Click the right arrow next to the VMnet1 settings and choose Subnet, as shown in Figure 3-28.

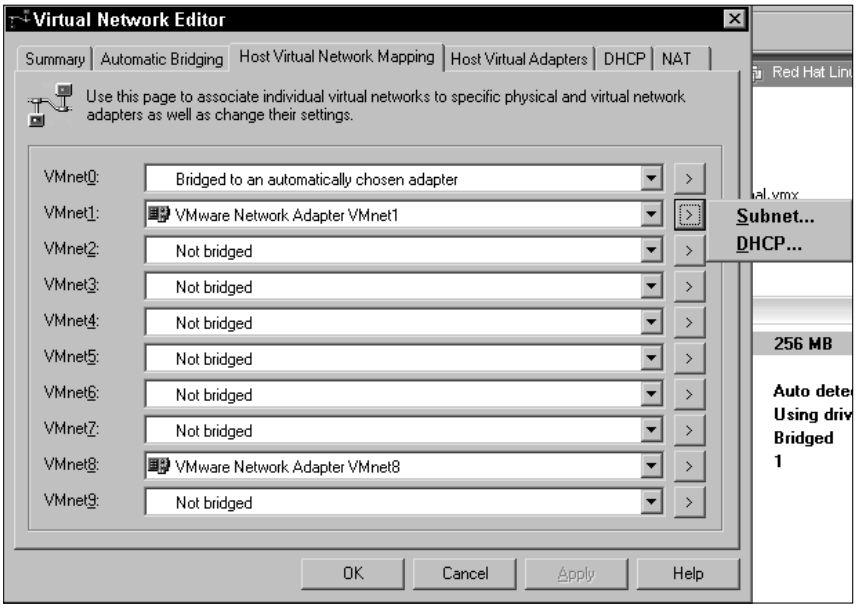

Figure 3-28

This will provide you with a window in which you can change your host-only network subnet, as shown in Figure 3-29. For this example, change it to 192.168.200.0 and click OK.

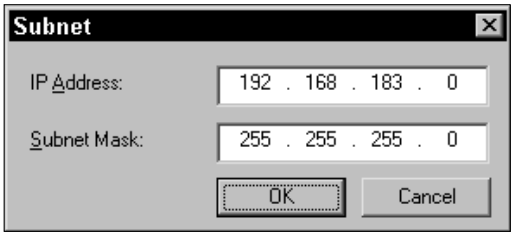

Figure 3-29

This alters the subnet in which the host-only network lives. Click the Apply button, and VMware Server automatically updates the DHCP server settings for VMnet1 to the new subnet. To take a look at what is available under the DHCP settings now (shown in Figure 3-30), just click DHCP from the fly-out menu shown back in Figure 3-28.

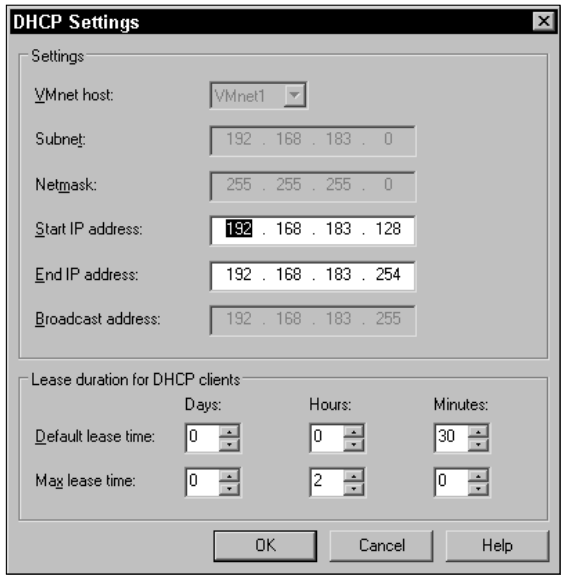

Figure 3-30

Notice that your DHCP settings adjusted to match your new subnet for VMnet1. Now, for most of your use the defaults for the DHCP server shown are more than adequate; however, there may be some special circumstances where they should be altered. For the most part VMware Server will correct any mistakes you make here like entering the wrong subnet address and so on. If you have a firm grasp of DHCP feel free to modify these to your needs.

Finally, take a look at VMnet8, the NAT network. Remember a NAT network shares the physical IP address of your host server's adapter with the machines connected to VMnet8 but issues them addresses from VMnet8's subnet via DHCP and forwards traffic to them. A discussion on how exactly Network Address Translation works and the advantages of it are beyond the scope of this book. Realize, however, that this type of network for your virtual machines is the most difficult to set up and could result in difficulties in communication with machines on your network. I suggest you use NAT only if you have a unique situation that requires it, like your host having only a single IP address to share as in a DSL or cable broadband connection.

Select NAT from the fly-out menu beside VMnet8 to open the window shown in Figure 3-31. This virtual adapter also has the subnet and DHCP menus that function identically to the previous host-only network setup.

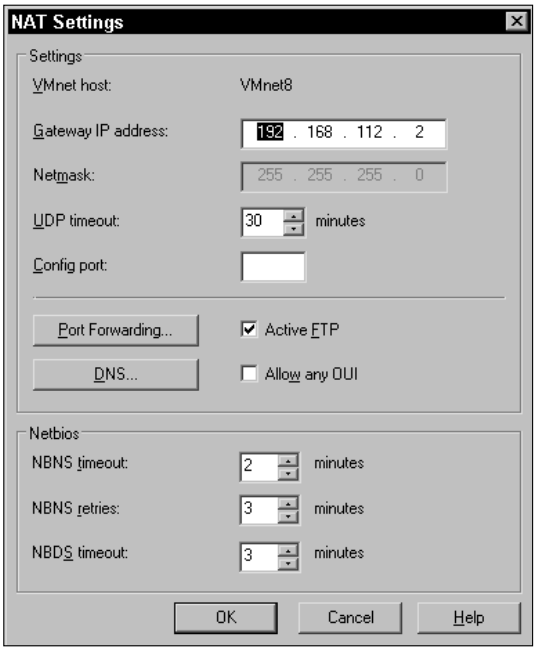

Figure 3-31

The Gateway IP address shown points to the virtual network adapter that was installed on your host server. You will see this in more detail on the next tab, Host Virtual Adapters.

The last two sections, Port Forwarding and Netbios settings, I will touch on only briefly:

- The Port Forwarding button allows you to forward TCP and UDP ports to specific machines behind the NAT. For example, you are unable to communicate from outside the NAT with a specific mail server that resides on TCP port 25 that is on a machine behind the NAT without first setting up the port forwarding in the NAT configuration. Without that information the NAT simply doesn't know what machine the traffic is meant for, and the traffic is just dropped. This occurs because the NAT machines are all exposed to the outside network by a single IP address. If you have three machines behind the NAT, as far as the outside network is concerned there is only one IP address so only one machine. The NAT itself must translate this information and forward the packets on to the proper server.
- The Netbios timeout settings are used only in the most special of circumstances. For that reason I will not go into them here. If you have further questions please consult the help documentation installed with VMware Server.

There can be only one NAT device on the host server.

Host Virtual Adapters

Next take a look at the *Host Virtual Adapters* tab, as shown in Figure 3-32.

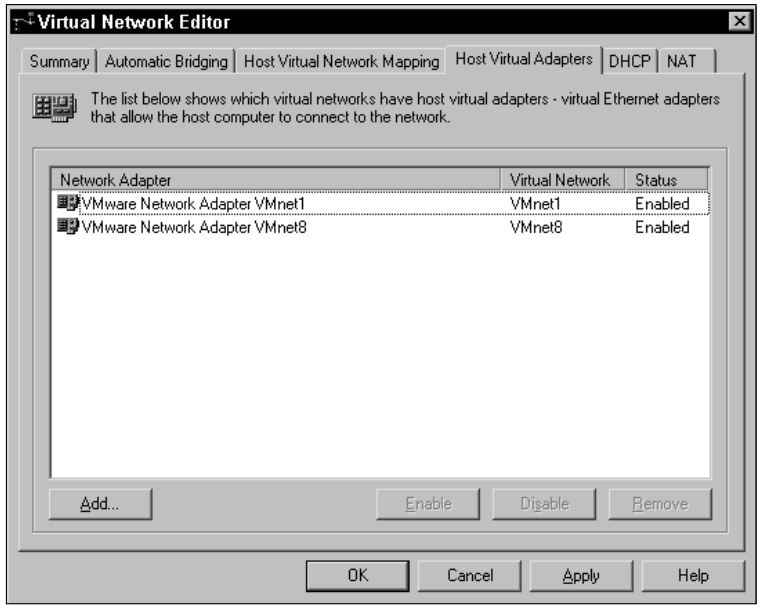

Figure 3-32

This is a list of the virtual adapters installed on your host. If you go into Network Connection on your Windows server you will find these adapters along with any physical adapters your host has. These are the gateway devices that the host-only and NAT virtual networks talk to. To add a new virtual network, other than bridge, you have to create a new virtual adapter in this tab. Clicking Add in the bottom corner will prompt you to choose a virtual network node (VMnet) that is free to create the new adapter on. Once you make this selection and click Apply, go back to the Host Virtual Network Mapping tab shown in the preceding figure to adjust the subnet and DHCP setting for the new network device. You can have up to nine different virtual networks on your host server.

More often than not the default virtual networks set up by VMware Server are plenty; however, there are some circumstances in which additional virtual networks might be useful. One such example is the isolation of a testing environment. For instance, if you are developing a software product that needs to interface with the Windows Server 2003 Active Directory, you obviously do not want the testing scenario to use a domain server on your production network, so you install a Windows Server 2003 domain inside a virtual machine. In order to keep things separate, you may want to consider isolating the virtual network in a way that places it on a different subnet than the production environment. This can be accomplished by configuring a new virtual network adapter.

DHCP and NAT

I'm not going to show the final two tabs, DHCP and NAT, in great detail here, but you can explore them on your own. Their basics are as follows:

- The DHCP tab provides a list of the configured DHCP servers for your virtual networks as well as a method to restart the VMware DHCP server.
- The NAT tab offers another way to configure your NAT network as well as assign the NAT networking device to another virtual network other than the default VMnet8. You can also cycle the NAT service from its tab.

Configuration of your virtual networks on Linux requires the use of the `vmware-config.pl` script located in `/usr/bin`. You must `su` to root to execute this script. Refer to the Chapter 2 discussion of installation on Linux for a run-through on what you will see while configuring the virtual networks on your host. One item to note is if you want to provide a specific subnet address for your virtual host-only or NAT network, choose *no* when the script requests to search for an empty subnet address.

Virtual Machine Security

As I have pointed out several times in this chapter, the security of your virtual machines and who can operate them is almost completely dependent on their file-level security in the host operating system.

Options for User Access Levels

Virtual machines by default are created to be private. That means only the user that created them can add them to their inventory and interface with them. In order to allow multiple users to access a single virtual machine, some modifications need to be made to their file-level security. You'll no doubt remember from earlier in the chapter the four ways a user can interact with a virtual machine:

- Browsing
- Interacting
- Configuring
- Administering

Each of these methods requires a different set of permissions to be granted on the virtual machine's files. Many of these permissions are based solely on the user's access to the virtual machine's configuration file (`.vmx`) located in the virtual machine's directory on the host. The following list steps through the four levels of access listed and the permissions needed to achieve them.

The .vmx Configuration File

Every virtual machine created with VMware Server possesses a single configuration file with the extension of `.vmx`. This file is made up of several key/value pairs of configuration information ranging from the name of the virtual machine to the Ethernet MAC address for the virtual network adapter.

Although this file is human readable and presents an easy to edit style, there are only a few very special cases where you should consider editing the configuration file by hand. The settings present in the configuration file can all be changed via the virtual machine configuration settings present in the VMware Server Console.

- **Browsing** to a virtual machine means you are allowed to connect to it from the console and view its power state, either on, off, or suspended, but that is all. You cannot view or interact with the virtual machine at all, and if the machine is powered on, the display in the console will remain blank. This level of permissions requires the user to be granted *read* permissions on the virtual machine's configuration file (`.vmx`) in Windows or the `r` permission on the virtual machine's directory in Linux.
- **Interacting** with a virtual machine requires the *read and execute* permissions for the user on the virtual machine's configuration file (`.vmx`) in Windows and the `rx` permission on the virtual machine's directory in Linux. A user with this level of access can change the power state of a machine, interact with its display, and connect and disconnect already configured devices from the machine. You cannot add new virtual hardware or change the virtual machine's configuration at this level. This is a safe level to put most of your users at to ensure the environment you set up stays static.
- **Configuring** the virtual machine requires a user to have *read, write, and execute* permissions for the virtual machine's configuration file (`.vmx`) in Windows and the `rwx` permission to the virtual machine's directory in Linux. This level of access allows everything stated in the previous points as well as the ability to add, remove, and configure the virtual machine and its hardware.
- **Administering** the host requires administrative privileges on the host server in both Windows and Linux. This is accomplished by either using an account in Windows that is a member of the local administrators group or using the root account in Linux. This level of access allows everything stated in the previous points as well as the configuration of the host settings. You should think carefully before allowing your users this level of access.

Configuring User Access Levels

There are various ways you can accomplish access control on Windows and Linux, but I want to offer a few suggestions to help you get started. If you have a large user base that will be accessing your virtual machines, you might want to look into this level of control. However, if your base is small, this level of control might hinder more than it helps.

Configuring User Access on a Windows-Based Host

On a Windows-based host I recommend you create three new security groups under local users and groups:

1. Start by right-clicking My Computer and choosing Manage. From there select Local Users and Groups.

   If your VMware Server is a member of a domain please check with your domain administrator on how they would prefer you set up these groups.

2. In the group folder right-click and add three new groups named *vmware-browse*, *vmware-use*, and *vmware-configure*. Because administering the host requires local administrator rights, you won't create a special group for that.
3. Once the groups are created, open My Computer and drill down to the parent folder where your virtual machines are located. The default is `C:\Virtual Machines`.
4. Right-click this folder, choose Properties, and select the Security tab.
5. From here you want to add your three new groups, so click the Add button and select the first group, *vmware-browse*. Click OK to add the group to the list.
6. Once it is added, ensure it is selected in the upper pane and adjust its permissions below. You want *read* and *list folder contents* checked and nothing else.
7. Now, repeat this process until all three groups are added, with *read and execute*, *list folder contents*, and *read* checked for the *vmware-interact* group, as shown in Figure 3-33, and *read and execute*, *list folder contents*, *read*, and *write* selected for the *vmware-configure* group.

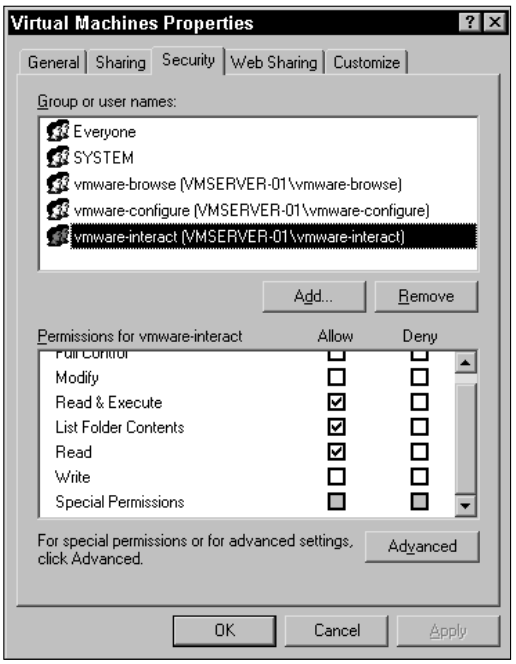

Figure 3-33

8. By default the settings will propagate down to the underlying virtual machine directories. Once this process is complete, return to the Local Users and Groups window and add the appropriate users to the group that matches the level in which they need access.

If you have a single virtual machine that someone needs higher permissions on than their group allows, you will need to stop the propagation of permissions for that specific virtual machine directory from the Advanced tab in file security and adjust the permissions for that user by hand. Remember, Windows permissions default to the most restrictive, so if you grant a user full rights to a machine but they are also in the `vmware-browse` *group, the most restrictive rights will take priority.*

Configuring User Access on a Linux Host

User permissions on Linux take on an entirely different form. First off Linux does not allow more than one group permissions to a file or directory at a time. Unlike the preceding Windows example where you had three different groups for your roles, a Linux-based host is incapable of this, so you will not be able to have as granular of a control system in place. There are, however, a few things you can do.

First, VMware Server runs as SUID root on a Linux host. This means that the `vmware` process runs as the root user and as such is afforded root access to the host system. This reality mandates you place at least some sort of access control on your virtual machines.

I recommend you create a `vmware-users` group on the Linux host and add the users to it you want to be able to use the virtual machines. You can do this by executing the following commands:

Just a reminder: All Linux-based procedures outlined in this book utilize Red Hat Enterprise Linux 4.0 WS with the 2.6.9-34.EL kernel.

1. Start by opening a console window and switching users (`su`) to root:

```
su -
```

2. Next you want to create a new group called `vmware-users`. This can be accomplished on a Red Hat server using the following command:

```
groupadd vmware-users
```

This will create your new group.

3. Next you need to add users to it. Repeat the following command for all users you wish to add to the `vmware-users` group. (Again, this is the command you will use on a Red Hat-based server.)

```
usermod -G vmware-users eric
```

4. Now, you need to adjust the permissions on the default location for your virtual machine files and use the `setguid` sticky bit.

By default on Linux any new file or directory created takes on the user and the group that the user who created the file belongs to. In order to maintain your `vmware-users` *group throughout all of your virtual machines you need to use the* `setguid` *option. This will force the server to keep the default group of* `vmware-users` *for all new files. This bit will be set by utilizing the* `chmod` *command in the following steps. The first three numbers immediately following the* `chmod` *command indicate levels of permissions from user, group, and world. The fourth number is the* `setguid` *bit.*

Change directories to the default location for your virtual machines. This was set by the install as `/var/lib/vmware/Virtual Machines` unless it was changed. If so, change to the new location and insert it in the following command line instead:

```
cd /var/lib/vmware/Virtual Machines
```

5. Now, you need to adjust the `Virtual Machines` directory to your new group. Enter the following command to do so:

```
chmod -R root.vmware-users .
```

This will modify the `Virtual Machines` directory to belong to the `vmware-users` group. The `-R` switch causes the change to propagate throughout its subdirectories.

6. Finally, you need to modify permissions for the parent and all subdirectories to allow members of the `vmware-users` group to create and modify your virtual machines as well as set the `set-guid` bit to keep your `vmware-users` group for all newly created files:

```
chmod 2770 .
```

This will change the permissions for the directory to *rwx* (read, write, and execute) for the user and *rwx* for the `vmware-users` group. All users that are not a member of the `vmware-users` group have no permissions.

7. If you require more granular control of your user's permissions on Linux, you will need to modify permissions on a case-by-case basis per virtual machine. For example, if you only want users to be able to interact with a particular virtual machine you will need to change into the directory for that machine and enter the following command:

```
chmod 2760 .
```

This will remove the *w* (write) permission for the `vmware-users` group and allow those users to only interact with the specific virtual machine.

Summary

This chapter concentrated on giving you a brief overview of the VMware Server Console and provided you with the information required for the chapters to follow. This chapter covered the following information:

- ❑ Getting to know the server console
- ❑ Host settings and console preferences
- ❑ Virtual machine settings
- ❑ Virtual hardware
- ❑ Virtual networking
- ❑ Virtual machine security

The next chapter steps you through creating your first virtual machine. This will be your first base image in your library. I walk you through configuring the virtual machine and packaging it for distribution in your testing and development environments.

4

Creating Your First Base Image

With the preliminaries of installation and configuration out of the way, this chapter walks you through the creation of a simple base image to get you started. It covers creating the virtual machine and configuring the hardware, installing the operating system, and using Sysprep, in the case of a Windows-based guest, to freeze and package your base image. When you are done with this chapter, you will be well on your way to creating an all-encompassing base image library. Because of this preliminary groundwork, the process of creating new test and development images will be as simple as copying over a base image, renaming, and starting it up.

What Is a Base Image?

Quite simply, a *base image* is a default guest machine used solely for the setup of other guest machines. Typically this is a virtual machine configured with an operating system already installed. Through the use of base images, you can harness the power of VMware Server as a rapid deployment test and development environment.

As an example, a base image could be a guest machine with a default install of Windows XP. You will not use the base image as a guest but simply make a copy of it for use as a production guest machine. This allows you to quickly provision new guests without the time required to set up and install an operating system. Through the creation of a thorough base image library you can expedite the creation of new machines to fit any situation.

With the creation of multiple base images comes the issue of licensing. Although I cannot cover all types of licensing, I will mention that you need to ensure you have the proper type and amount of licenses available for each base image you create. Please consult your End User License Agreement (EULA), the Microsoft web site for volume licensing, or your licensing representative for the operation system in question prior to creating your base images. In most instances the use of base images requires at least some sort of volume licensing agreement or equivalent for the operating system in question.

Creating Your First Virtual Machine

First off you need to create a new virtual machine. This can be accomplished via the New Virtual Machine Wizard I mentioned back in Chapter 3. The wizard offers a multitude of choices for hardware and machine configuration, but does a decent job of configuring the defaults based on the operating system selected. I want to take the time now to walk you through an advanced virtual machine setup step by step. I will be using a Windows-based VMware Server and setting up a Windows XP Professional virtual machine as your first base image.

1. You can create your first base image from either the server itself or from a remote client. Start by opening the VMware Server Console and selecting the Home tab in the main pain. Once there, click the New Virtual Machine icon to open the wizard shown in Figure 4-1.

New Virtual Machine Wizard
Welcome to the New Virtual Machine Wizard
This wizard will guide you through the steps of creating a new virtual machine.
VMware Server
CONSOLE
< Back
Next >
Cancel

Figure 4-1

This wizard will guide you through the entire creation process. Go ahead and jump right in.

2. If you click Next, you will be asked if you want to create your new virtual machine using a typical configuration or a custom one. The difference here is that VMware has a default set of hardware items and settings for each type of operating system in the list. For example, a typical

virtual machine setup for Windows Server 2003 is to use SCSI virtual disks, whereas the typical setup for a Windows XP-based machine is to use an IDE virtual disk. By choosing typical you are allowing VMware to set up your hardware based on its predetermined defaults. Now, this isn't a bad thing because for the most part the default setup is perfectly fine. However, you are provided with the flexibility to customize the setup during the creation process. Figure 4-2 shows your choices.

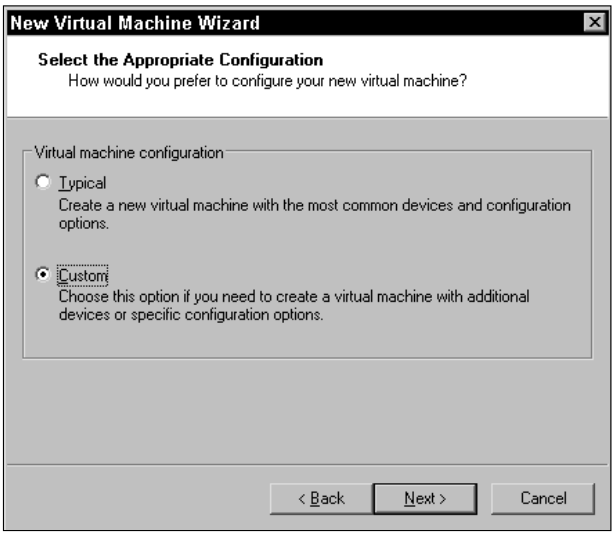
New Virtual Machine Wizard
Select the Appropriate Configuration
How would you prefer to configure your new virtual machine?
Virtual machine configuration
Typical
Create a new virtual machine with the most common devices and configuration options.
Custom
Choose this option if you need to create a virtual machine with additional devices or specific configuration options.
< Back
Next >
Cancel

Figure 4-2

In the interest of learning choose *Custom* and click Next. The subsequent screens will allow you to completely configure almost every aspect of your virtual machine and its hardware.

3. In Figure 4-3 you will see you need to select the intended operating system for this base image. The selection here is not really that important. You should already have an idea of what this base image is for so making the right selection is easy; however, this really boils down to what defaults the wizard will provide you in the upcoming steps and not the functionality of the virtual machine itself. Don't worry if you change your mind later. Like most settings in the upcoming steps, you can always adjust them later if your needs change.

   As you can see in the preceding figure you have the basic groups of operating systems presented as well as a drop-down menu for more specific choices within those groups. For the purpose of this example, you will create a Windows XP Professional base image. Select Microsoft Windows and choose Windows XP Professional from the drop-down list.

4. Now VMware needs to know where to put this new machine. Figure 4-4 shows the next step. You are prompted to provide a name for your new machine as well as the location to place its files. I recommend because you are creating a base image that you name the virtual machine something other than the default. I use a specific naming scheme that I introduce in Chapter 6, but for now name this image `base-winxppro`. In Chapter 6 you'll start to realize why I choose this scheme and its benefits. When finished, click Next.

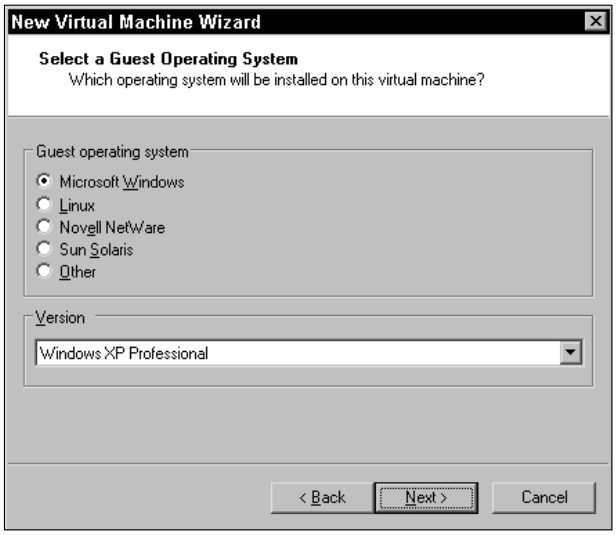

Figure 4-3

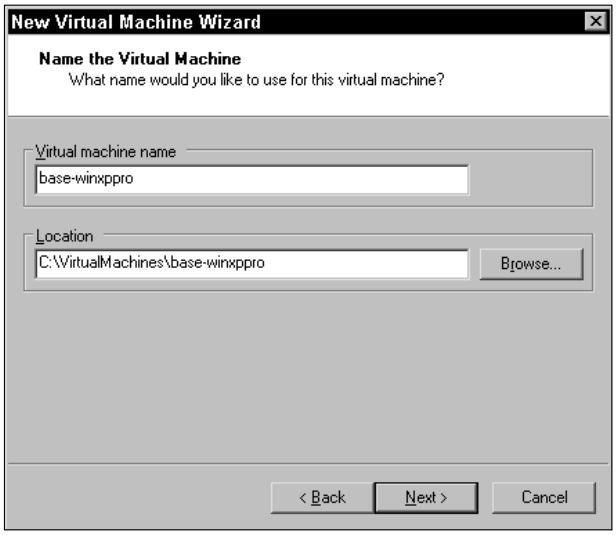

Figure 4-4

5. Next you will find yourself at the screen shown in Figure 4-5, setting access permissions.

   As you may recall I covered basic virtual machine security in Chapter 3. By default all new virtual machines are created as private. This means that only the user under which the new virtual machine was created can add it to their inventory and power it on. For now leave the default setting of private and continue on.

Basic Virtual Machine File Extensions

While I am on the subject of where to place the virtual machine files, it is a good time to go over the basic file extensions and purpose of the files you will find in your virtual machine's directory.

As I discussed in preceding chapters, a virtual machine simply consists of a series of files within a folder unique for that specific machine. The chances of your having to edit or manipulate these files directly is remote, but having a grasp of their function helps in understanding the operation of each virtual machine.

Unless otherwise noted all the following extensions are prefixed by the name of the virtual machine.

- **.vmx** — This file stores the configuration data for the virtual machine. It is in plain text format and can be edited directly as long as the virtual machine is not powered on. Changes in the virtual machine's configuration via the server console are directly reflected within this file.
- **nvram** — This file stores the current configuration of the virtual machine's BIOS. This file should not be modified and exists only while the virtual machine is powered on.
- **.vmdk** — This is the virtual disk file. Your virtual machine may have multiple `.vmdk` files depending upon its virtual disk configuration.
- **.log** or **vmware.log** — These files contain logged events that take place during the operation of your virtual machine. You can utilize the information in these files for troubleshooting if something goes wrong or provide them to VMware Support upon request. The log files exist only after the virtual machine is powered on for the first time and are safe to delete when the virtual machine is powered off if you so choose.
- **.vmdk.REDO** — A `REDO` file is created when your virtual machine has either a snapshot or the virtual disk is in nonpersistent mode. This file contains all the changes made to a virtual disk while the machine is powered on. It in addition to the `.vmdk` file is required for proper operation of your virtual machine. There will be a unique suffix appended to all `REDO` file extensions to ensure they are unique.
- **.vmss** — This file stores the state of the virtual machine while it is suspended. When you suspend a virtual machine, the current memory state and other critical information is stored within this file. Bringing a machine out of suspended mode will load this file and return the machine to its exact state prior to suspension.
- **.vmsn** — This file stores the state of your virtual machine at the time a snapshot was taken.
- **.vmx.sav** — When you take a snapshot the current hardware configuration must be captured because any change in virtual hardware will render a snapshot ineffective. This file stores the virtual machine's hardware configuration at the time the snapshot was taken.

6. The next step is present only for Windows-based VMware Server installations and involves the startup and shutdown options for the new virtual machine, as shown in Figure 4-6.

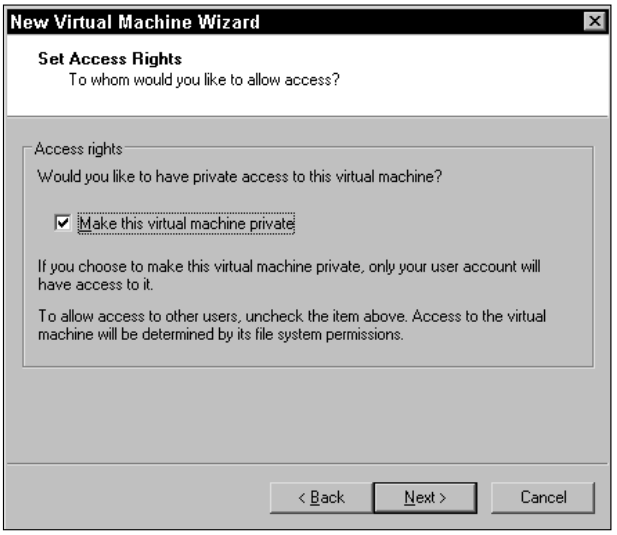
New Virtual Machine Wizard
Set Access Rights
To whom would you like to allow access?
Access rights
Would you like to have private access to this virtual machine?
Make this virtual machine private
If you choose to make this virtual machine private, only your user account will have access to it.
To allow access to other users, uncheck the item above. Access to the virtual machine will be determined by its file system permissions.
< Back
Next >
Cancel

Figure 4-5

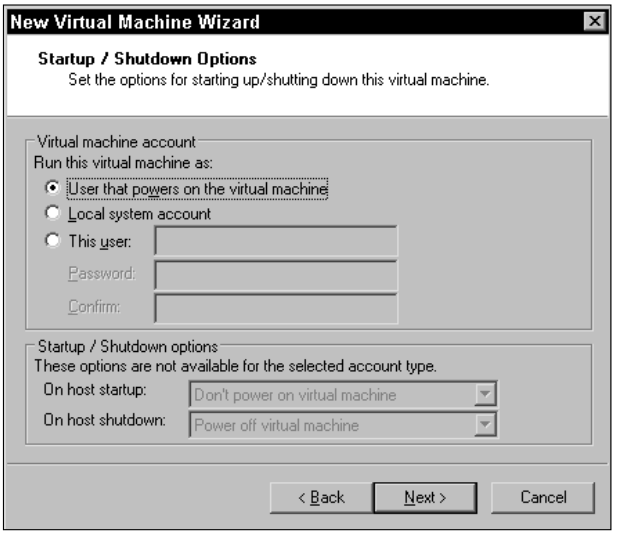
New Virtual Machine Wizard
Startup / Shutdown Options
Set the options for starting up/shutting down this virtual machine.
Virtual machine account
Run this virtual machine as:
User that powers on the virtual machine
Local system account
This user:
Password:
Confirm:
Startup / Shutdown options
These options are not available for the selected account type.
On host startup:
Don't power on virtual machine
On host shutdown:
Power off virtual machine
< Back
Next >
Cancel

Figure 4-6

You have three choices here:

- By default your new virtual machine, once powered on, will run under the security context of the user that powered it on. If this user is not a member of the server's administrative group, this can affect the machine's ability to access certain local hardware items like the USB controller. All users that connect to the virtual machine after it is powered on will be afforded only the access allowed by the user that powered it on.

Refer to Chapter 3 for more information on the security rights needed for specific local hardware access.

- The second choice is to run the virtual machine under the local system account. This account by default on Windows-based machines runs under a local administrator context. You are allowed to enable this choice only if you are currently logged in to the console as a user with administrative rights on the VMware Server.
- Finally, you can choose a specific user to run the machine under. This user can be a local user, a local administrator, or a domain account if your VMware Server is part of a Windows domain.

The startup and shutdown options are made available when you choose either the local system account or a specific user that has at least local administrator rights. These options control whether or not a machine is powered on along with the host server and shut down prior to the host server's shutting down.

Refer to Chapter 3 for some additional settings required, such as the time delay for startup, prior to setting this option.

7. Once that step is complete you can move on to the Processor Configuration, as shown in Figure 4-7.

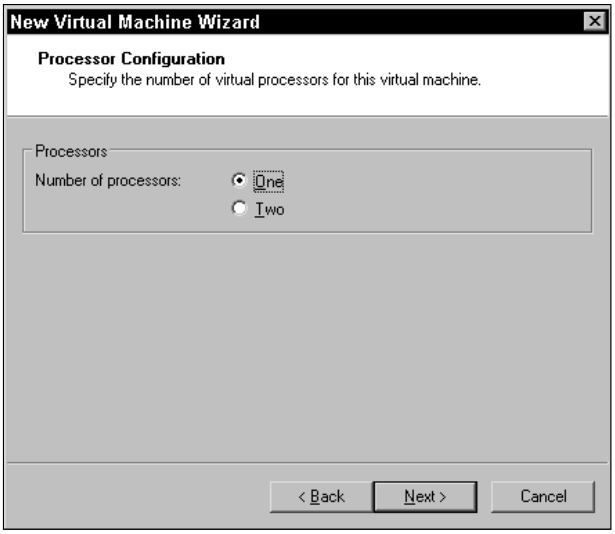

Figure 4-7

This allows you to set how many processors you will make available to this virtual machine. VMware Server itself only supports a host with a maximum of two physical processors installed. For that purpose VMware Server considers the following types of processors to be a dual processor:

- A single processor with hyper-threading enabled
- A single dual-core processor
- Two processors, neither of which have hyper-threading nor are dual core

When running a virtual machine with two processors configured on a server that has only hyper-threading, you will experience less than optimal performance, and therefore I recommend you set up multiple processors only when your server has a dual core or multiple processors. You can run multiple virtual machines each configured with two processors.

8. Now you are faced with probably one of the most important configuration items, memory. As shown in Figure 4-8, you need to select the amount of memory to reserve for your virtual machine.

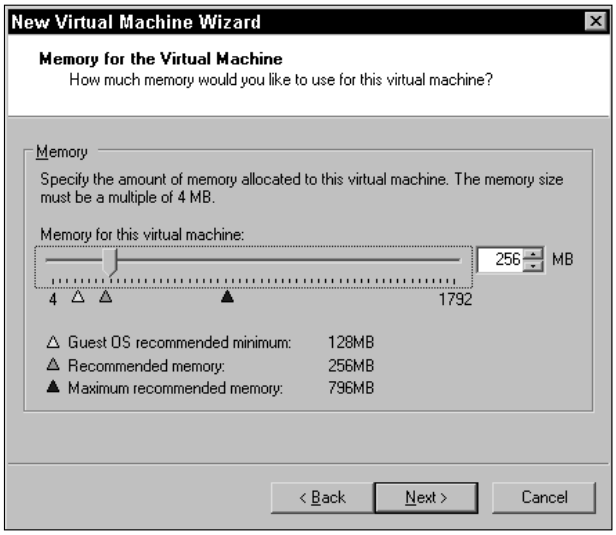
New Virtual Machine Wizard
Memory for the Virtual Machine
How much memory would you like to use for this virtual machine?
Memory
Specify the amount of memory allocated to this virtual machine. The memory size must be a multiple of 4 MB.
Memory for this virtual machine:
256
MB
4
1792
Guest OS recommended minimum: 128MB
Recommended memory: 256MB
Maximum recommended memory: 796MB
< Back
Next >
Cancel

Figure 4-8

Depending on the selection you made for the type of guest operating system to be installed on your new virtual machine, the VMware Server will provide you with a default memory value. More often than not this default value is well below what you will need to have a smooth experience. The slider control will provide a range between the lowest possible selections to the maximum memory available for virtual machine usage on your server. You will notice the maximum memory is not the actual total memory available on your server. This is because a small amount of memory is reserved for the overhead of running the guest machine. Only experimentation will help you discover the amount of memory you want to give your guests. I recommend you provide a Windows XP guest like that in the example no less than 512MB.

You are restricted in the amount of memory you can reserve for a virtual machine based upon the file system present on the host server. FAT16-based file systems allow a maximum of 2GB of memory to be configured. FAT32-based file systems still only allow a maximum of 2GB of virtual memory even though they can have a file size up to 4GB. If you configure more than 2GB, the machine will refuse to power on. This is due to how VMware Server stores the virtual memory files on the host file system. Other file systems do not possess this limitation.

9. In Chapter 3 I discussed virtual networking, the options available, and the configuration of new virtual networks. Your next step is to choose what type of virtual network you desire for the new virtual machine. Figure 4-9 shows the options available.

   Select the type of virtual network you desire. Refer to Chapter 3 if you need help determining which type of virtual network to select. If you defined a special virtual networking setup, you will have to select it for the virtual machine's use after the wizard has completed.

10. Next is where you select the type of adapter architecture you wish to provide to your virtual machines, as shown in Figure 4-10. An IDE adapter is always ATAPI and cannot be changed; however, you have two choices for the SCSI type.

    Depending upon the choice you made for the operating system you are targeting with this virtual machine, either BusLogic or LSI will be selected by default.

    - Newer operating systems will default to the BusLogic adapter, and they typically have support built in for them.
    - The LSI adapter offers better performance; however, it usually requires you to download the specific driver from the VMware web site (`www.vmware.com`) to use. Windows Server 2003 now provides the LSI driver standard and therefore is the default selection for virtual machines targeted to that platform. Select drivers and tools from the VMware Server product page to download the vmSCSI drivers.

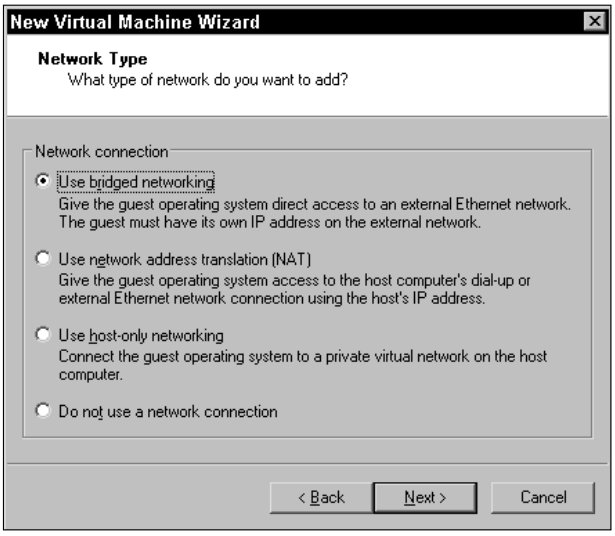
New Virtual Machine Wizard
Network Type
What type of network do you want to add?
Network connection
Use bridged networking
Give the guest operating system direct access to an external Ethernet network. The guest must have its own IP address on the external network.
Use network address translation (NAT)
Give the guest operating system access to the host computer's dial-up or external Ethernet network connection using the host's IP address.
Use host-only networking
Connect the guest operating system to a private virtual network on the host computer.
Do not use a network connection
< Back
Next >
Cancel

Figure 4-9

The selection here does not require you to use a SCSI virtual disk for your machine.

Unlike most settings in the New Virtual Machine Wizard, the selection of the SCSI adapter type cannot be changed after the machine is created.

11. Next you need to select the type of virtual disk to set up for your virtual machine, as shown in Figure 4-11. You will almost always select the default and create a new virtual disk, and as such, I will not cover the other two choices. Refer to the VMware Server documentation for information on how to use a physical disk or partition for your virtual machine.

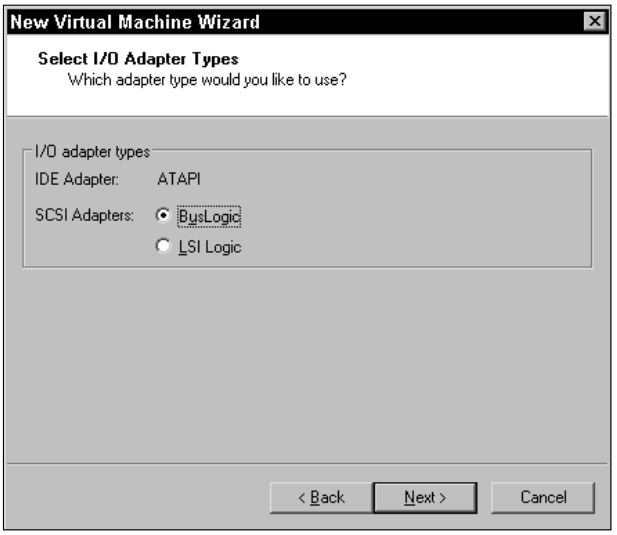

Figure 4-10

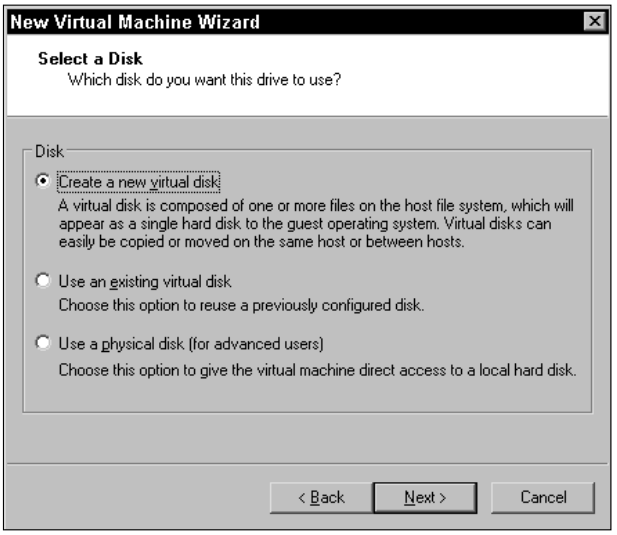
New Virtual Machine Wizard
Select a Disk
Which disk do you want this drive to use?
Disk
Create a new virtual disk
A virtual disk is composed of one or more files on the host file system, which will appear as a single hard disk to the guest operating system. Virtual disks can easily be copied or moved on the same host or between hosts.
Use an existing virtual disk
Choose this option to reuse a previously configured disk.
Use a physical disk (for advanced users)
Choose this option to give the virtual machine direct access to a local hard disk.
< Back
Next >
Cancel

Figure 4-11

Because in this example you are creating base images that you will later freeze and compact for easy storage and distribution, creating a file-based virtual disk is your best choice. The default selection provides that.

12. The next step continues on with virtual disks. Figure 4-12 shows your choice for the virtual disk architecture you want to utilize. VMware Server will default the selection based upon the target operating system you selected at the beginning of the wizard. You are not restricted to using the default provided; however, in most cases the wizard is correct and the default will provide you with the best solution.

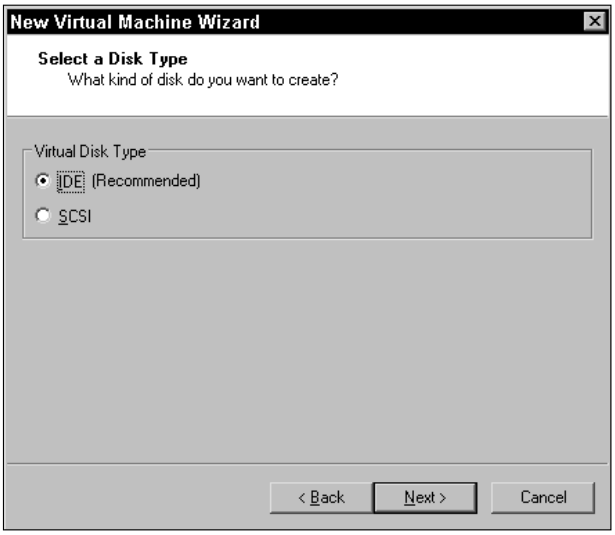

Figure 4-12

You will notice that in the example for Windows XP the wizard defaulted to an IDE disk. Having created numerous Windows XP base images, I can tell you this is sufficient. For example, however, if you had selected Windows Server 2003 as the operating system, the default here would have been SCSI.

13. Next and probably the second most important configuration item after memory is the disk size, as shown in Figure 4-13.

Here you have to consider a variety of factors based on the operating system you are installing plus any applications you may add to the virtual machine later. Once the disk size is set you cannot change it via conventional methods, such as the console configuration settings. Part of what will help you determine how large to make the virtual disk file is if you want to allocate the disk space now or have it expand as needed. Keep in mind that you can always add an additional IDE or SCSI virtual hard disk at a later time to increase space. Any additional drives appear to the virtual machine as a separate disk.

VMware Server by default chooses to allocate the disk space when you first create the virtual machine. This offers a slight performance boost because it does not have to allocate and expand your virtual disk as items are installed. However, though this offers a slight performance boost, it also takes up a great deal of your disk space. A virtual disk that is allocated at 20GB will create a file 20GB in size from the very beginning regardless of how much actual space is occupied on the disk itself. You will need to weigh the advantages out to meet your specific requirements.

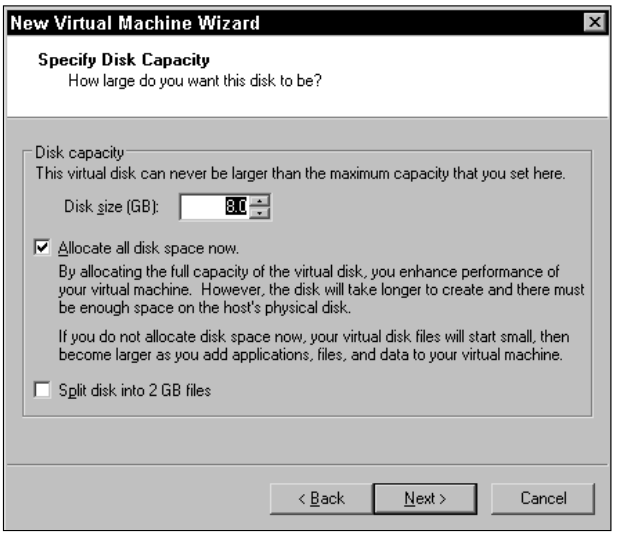
New Virtual Machine Wizard
Specify Disk Capacity
How large do you want this disk to be?
Disk capacity
This virtual disk can never be larger than the maximum capacity that you set here.
Disk size (GB):
8.0
Allocate all disk space now.
By allocating the full capacity of the virtual disk, you enhance performance of your virtual machine. However, the disk will take longer to create and there must be enough space on the host's physical disk.
If you do not allocate disk space now, your virtual disk files will start small, then become larger as you add applications, files, and data to your virtual machine.
Split disk into 2 GB files
< Back
Next >
Cancel

Figure 4-13

In terms of a development image where your team will be using the guest machine for everyday work, a pre-allocated disk may offer you greater performance and disk space could be less of an issue. However, in the case of base images for testing purposes, the actual performance may not be as important as the actual portability and ease of storage for the virtual machine. I recommend you do not pre-allocate disk space in any base image that will be used sporadically or for which you may require several different copies to exist at once. I have virtual machines utilizing both pre-allocated and expanding disks, and the performance difference between the two is hardly noticeable.

Further, if you are using a file system based on FAT16 or FAT32 you should select *Split disk into 2GB files*. This is because of the maximum file size allowed on those file systems.

14. This brings you to the final step, as shown in Figure 4-14, where to store the new virtual disk.

This simply is asking for the location to store the virtual disk file and its name. The default is to place it into the same directory as the virtual machine's configuration file. The only requirement here is that the virtual disk be in a location accessible by the users needing to utilize the virtual machine. One thing to note is that the default name for the virtual disk file is the same as the guest operating system selection. I recommend you rename the virtual disk file to match the name of the virtual machine. This facilitates easier organization later when you have dozens of virtual machines lying around.

Once you click Finish, the New Virtual Machine Wizard is complete. You will notice a new virtual machine is listed in the inventory and its tab is selected in the main pane. From this point you need to install an operating system onto the new virtual machine.

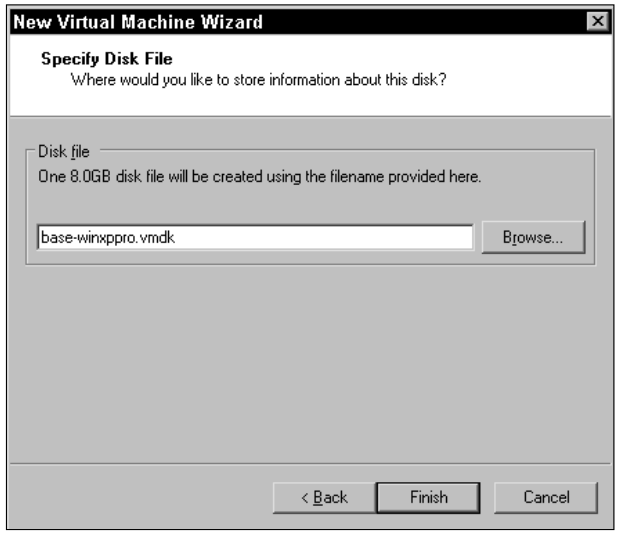

Figure 4-14

Installing a Guest Operating System

Installing an operating system on your guest is remarkably similar to installing an operating system on a physical computer. You simply insert the operating system CD into the drive and turn on the guest machine.

Once you are out of the New Virtual Machine Wizard, you should have the tab selected in the VMware Server Console for the new virtual machine you just created. Take a look at your hardware configuration, specifically the CD-ROM drive. Start by selecting Edit Virtual Machine Settings under the Command section on the new virtual machine's tab. This will open the hardware configuration screen for your virtual machine, as shown in Figure 4-15.

Now the preceding figure may not match your setup exactly, but it gives you an idea of where you are going. Go ahead and click the CD-ROM drive from the device list on the left. The right pane will change to show you the specific setting for your CD-ROM drive, as shown in Figure 4-16.

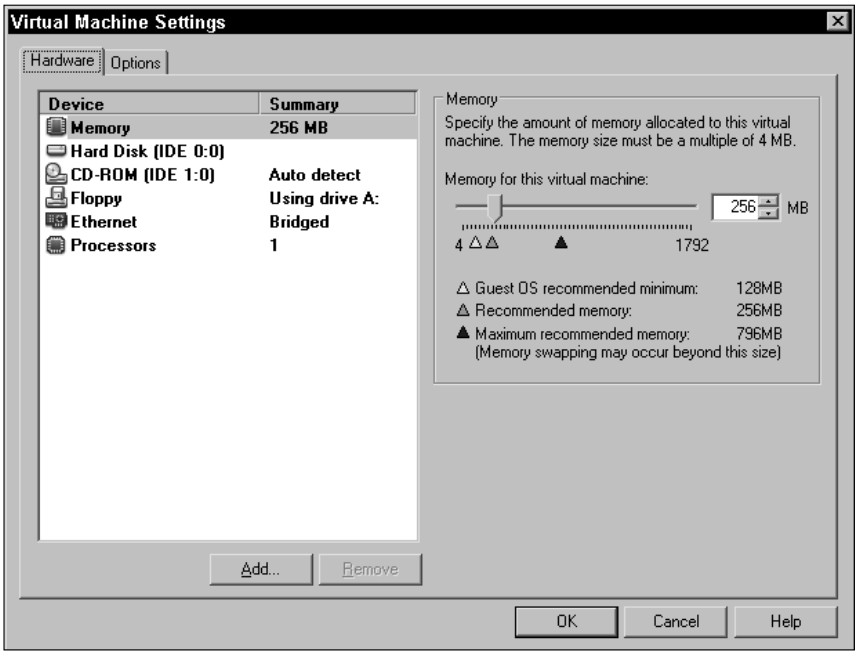

Figure 4-15

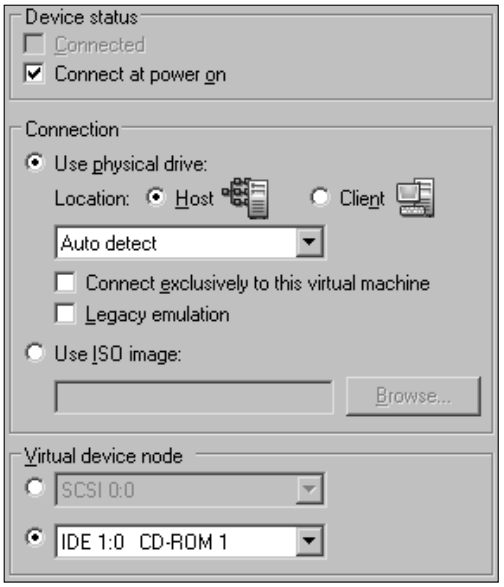

Figure 4-16

Right off you'll notice the CD-ROM drive is not set to be connected at power on. First make sure and place a checkmark under Device status for Connect at power on. Now, the beauty of this "disconnected" architecture VMware Server provides is that you can utilize the physical CD-ROM drive of either the host server or your local machine. There is one caveat, however, and that is you cannot select to use a physical drive on the client machine unless the virtual machine is powered on. Once the machine is powered on you can connect your local hardware via the VM menu provided on the console. For now move past that; I want to talk about specific drives on the host computer and ISO images.

An ISO image is simply a standard way to archive the entire contents of a CD-ROM into a single file in an uncompressed format. Many of your popular file compression utilities or CD Recording software allow you to make an ISO image of a CD-ROM. My personal favorite is a program called WinImage. You can download this application in shareware format from `www.winimage.com`. This program not only allows you to create ISO images of CD-ROMs but also images of your floppies as well.

For more information on ISO files and how to create them, see Chapter 6.

VMware allows you to not just connect your machines to a physical drive but also to an ISO image file instead. VMware will emulate the CD-ROM drive for you. The guest machine will simply think there's an actual CD in the drive. The same holds true for your floppy drive. I personally do not create virtual machines with virtual floppy drives because I simply do not use them; however, you can create and use a floppy disk image just the same as an ISO image described previously.

So why am I telling you all this? Plain and simple, it boils down to performance. Although you can install your base operating system off a physical CD-ROM disk, you can obtain a significant speed increase if you install from an ISO image due to the simple fact that hard disk access is still faster than CD-ROM.

- To install the operating system from a physical disk make sure host is selected from the location radio button and drop down the list of drive choices. You may choose to leave *Auto detect* but I recommend you point your guest machine to a specific drive on the host server. Once you make your selection, click OK at the bottom of the settings screen.
- If you want to use an ISO image, select the radio button for *Use ISO image* and click Browse to browse to the location of your ISO image file. Once this selection has been made, click OK.

Now that you have either a physical CD-ROM drive configured with the disk inserted or you are utilizing an ISO image, it's time to power on your virtual machine. You can do this from the Commands section of the machine's tab, or from the toolbar by pressing the green play button. You will see the console window adjust, display the virtual BIOS window, and start the boot sequence, as shown in Figure 4-17.

The virtual BIOS is set by default to boot to CD-ROM if no bootable hard disk is found. If by chance you need to change this order, you can enter the BIOS setup screen by pressing F2 during startup.

If your physical drive or ISO image configuration is correct, you should see the operating system installation startup.

Local host - VMware Server Console
File Edit View Host VM Power Snapshot Windows Help
Home
base-winxppro
vmware®
Starting
Press F2 to enter SETUP, F12 for Network Boot, ESC for Boot Menu
You do not have VMware Tools installed.
VMware Server 1.0.0
Start
Local host - VMwa...
C:\Documents and ...
5:43 PM

Figure 4-17

Troubleshooting Guest Operating System Installation

Because of the vast number of possible operating systems there is no way to cover all troubleshooting options here; however, there could be a few hiccups along the way that I want to point out.

You can troubleshoot problems with your install just as though they were on a regular physical machine. If the virtual machine cannot find a suitable hard disk or CD-ROM to boot from, it will attempt to boot via the network utilizing `bootp`. If this happens, verify you have the operating system CD in the proper drive that you selected in the virtual machine setting and that the drive is set to connect at power on. If you look in the bottom-right corner of the console you will see a few iconic representations of major pieces of virtual hardware. If the icon for the CD-ROM drive has a small X through it, you need to enable it either through the virtual machine settings with the guest powered down or by right-clicking the icon and selecting *Connect*. Figure 4-18 shows you the hardware icons close up.

Figure 4-18

If you are using an ISO image file for your CD-ROM and your guest fails to boot, verify the ISO image is good by either opening it in your favorite file archiving utility or in the software you used to create the image. If all else fails, create a new image and try again.

Installing VMware Tools on Windows Guests

VMware offers a set of installable drivers and performance enhancers called VMware Tools. These should be installed on every single guest you create. They are required for proper video display, for the automatic resizing of the console and guest screens, and for the automatic grabbing and releasing of your mouse as it moves from console to client desktop. It also installs the network and disk drivers necessary for the operating system to interface with the virtual hardware. They are so important that the VMware Console will actually beg you to do it with big blue bold letters in the bottom right-hand corner of the console window.

After you complete the main operating system installation, leave the guest running and click the VM menu from the console and select Install VMware Tools from the menu as shown in Figure 4-19.

Figure 4-19

For this to work you need to have a configured and connected CD-ROM drive for your guest machine. The VMware Console will temporarily disconnect any physical drive or ISO image from your machine and mount its own image containing the VMware Tools package. Depending upon your operating system this may trigger the guest machine to automatically start the installation. Finish installing the VMware Tools package and reboot your guest. The VMware Console will restore your previous CD-ROM configuration automatically. Once the guest is back up, the console message to install the tools package should be gone.

Installing VMware Tools on Linux Guests

Installation of the VMware Tools package on a Linux-based guests can get a bit more complicated. The installation itself is somewhat simple but the nature of Linux leaves a lot of variation on how it is accomplished.

Like our Windows counterpart the first step is to select the Install VMware Tools option from the VM menu as shown in the preceding screenshot, Figure 4-19. After this is where it starts to vary between Linux distributions. Once you select the menu option, the ISO file containing the Linux VMware Tools package will be attached to your virtual CD-ROM drive. How this is handled by your guest is a whole different story. Some distributions detect a disk change event and automount all CD-ROMs as they are inserted. Yours may or may not be one of those, so I will cover what to do if it does not automount.

As is the case with all of the Linux examples, this one uses Red Hat Enterprise Linux WS.

VMware Tools must be installed using a text console. What I mean by that is you cannot just open up a new terminal window within your X environment. The typical hot key to do this is Ctrl+Alt+F1 through F6. Just pick a function key between 1 and 6 and you should switch to a text console login. It is not necessary to switch run levels; however, if you prefer, you may.

Sending hot key combinations to a guest machine running within a VMware Console may require you to first depress the VMware hot key Ctrl+Alt+Space. This hot key informs the console you are about to send a hot key combination to the virtual machine. Press Ctrl+Alt+Space, release Space but leave Ctrl+Alt depressed, and then follow with your function key. Obviously, if you have defined a different hot key modifier in the console, insert it in place of Ctrl+Alt.

Once you switch to a text console, go ahead and either log in as root or log in under a user account. If you log in under a user other than root make sure and issue the switch user command as follows:

```
su -
```

Now, after you select the Install VMware Tools option from the VM menu on the VMware Console Red Hat will actually mount the disk for you as long as X is still running in the background. If you switch run levels, this may not happen automatically. The question you are left with, however, is where exactly it mounted to. You can issue the `mount` command to get a list of currently mounted hardware devices or simply just remount it to a location you are aware of.

For now you can just mount your own copy. It seems a little easier than figuring out what your specific distribution is doing. First, you need to switch to the root directory:

```
cd /
```

Next you have to create a location to mount the CD-ROM drive to. Usually most Linux distributions have a `/mnt` directory just for this purpose. This, however, could also be `/mount`. Honestly, it doesn't matter where; you are just going to remove it once you are done. For now assume you have a `/mnt` directory already. If you do not, you can simply create one with the `mkdir` command. Change into the `/mnt` directory and make a place for your CD-ROM to mount to:

```
cd /mnt
```

Now you need a place within the `/mnt` directory to actually mount the CD-ROM to. You can execute an `ls` command to view the current contents of this directory or simply create a new directory to hold your temporary mount point. For this example, just create a new location; however, this directory may already exist. It depends entirely on the distribution you are using.

```
mkdir /mnt/cdrom
```

Now that you have a place to mount to, it is time to mount the drive. Issue the following command to mount the CD-ROM drive to the location you created `/mnt/cdrom`:

```
mount /dev/cdrom /mnt/cdrom
```

What you are looking for is a return message that says something to the effect of `mounting read-only`. Once you receive that message you are good to go, and you can continue on.

Most distributions have a temporary folder, usually called `/tmp`. This is where you are going to copy the VMware Tools archive file to so you can extract it and install. (Again it really doesn't matter where you put this file; it doesn't have to be in `/tmp`, so feel free to insert any other directory in the command that follows to fit your needs.) Change into the `/tmp` directory, copy the VMware Tools archive file over, and then unmount your CD-ROM drive:

```
cd /tmp

tar zxf /mnt/cdrom/vmware-linux.tools.tar.gz

umount /mnt/cdrom
```

As long as all of that went well and you received no errors, you are ready to start installing the tools package into your Linux guest. Within the `/tmp` directory you should now have a `/vmware-tools-distrib` directory. Change into that directory and execute the installer script:

```
cd vmware-tools-distrib
./vmware-install.pl
```

In previous chapters I have taken the time to walk you through the questions the VMware installer asks, but in this case the install is short, simple, and all of the default values will work just fine. Continue through the installer and accept the defaults.

Once that is complete you will need to briefly configure the VMware Tools. Issue the following command in the console:

```
/usr/bin/vmware-config-tools.pl
```

Really all this is going to do is set up your video resolution within X. Select the desired resolution from the list and the configuration will be complete.

At this point I recommend you either completely restart the guest by issuing the `shutdown` command shown as follows or logging out of the text console, switching back to your X session, and restarting the X environment. You can determine which of theses options is easiest. If you want to just restart the entire machine, issue the following command:

```
shutdown -r now
```

If you want to just restart your X environment, log out of the root account on the console and switch back into your X session by pressing Ctrl+Alt+Space, releasing Space, and then depressing F7. This should bring the X session back up on your screen. At this point you can press Ctrl+Alt+Space, release Space, and then depress the Backspace key. This triggers the X session to reload and thereby reloads the VMware Tools settings you made for the resolution.

There's one final step here for the tools package on Linux. On a Windows guest there is a little VMware Tools icon present on the taskbar. To gain the advantages of the tools package, it needs to be running on the guest machine. You should make every effort to do this. It is quite simple on a Windows system because the installer takes care of it for you. The task can be a little more difficult on a Linux distribution. The simplest way is to search your distribution's documentation for startup programs. Red Hat distributions that use Gnome as their X windows manager can add programs to start automatically under Programs➪Settings➪Session➪Startup Programs. This, however, is not a solution for all distributions. Although it is not the most elegant solution you can always issue the following command in an X windows terminal:

```
vmware-toolbox &
```

This will start the VMware Tools package and push it to the background to run. After you issue that command, you can close your terminal window. Note, however, that this is not a one-time event. If you start it this way, you will need to do so every time you start up the guest. I recommend you consult your distribution's documentation on how to add programs to automatically start for each X windows session.

If you run into trouble performing the tools package install on Linux, you can always consult the VMware documentation or the VMware Server forums on `www.vmware.com` for more information.

Freezing Your Windows Guest with Sysprep

Sysprep, or the System Preparation Tool, is a utility from Microsoft to ease in the rapid distribution of Windows-based images. There is a specific Sysprep tool for every version of Windows and you should not mix them. The one specific to the version of Windows you have installed can be found on the original installation CD or ISO image under the `\Support\Tools` folder in a file called `Deploy.cab`.

> *New versions of Sysprep are also issued along with each service pack. If you have a service pack installed please visit the Microsoft web site and search for your service pack level, operating system, and Sysprep. In the case of Windows XP SP2, you can search for Windows XP SP2 Deploy Tools.*

Sysprep can be an incredibly complex tool to use so I don't want you to get lost in the documentation. As an example of what it can do, if you have ever bought a new computer from any of the big name companies and turned it on for the first time, it puts you into a semi-install routine for Windows and asks you a series of questions. This is done using mini-setup, a feature of Sysprep. Basically what you want Sysprep to do is freeze your image so that when you create copies and start them up for the first time it will enter this mini-setup phase and generate a unique security identifier, or SID, for the new guest machine. Without this step all of the copies of your Windows-based images would have the same SID, and this can cause you serious problems if the machines ever have a chance to actually talk to one another over the network.

Because I am using Windows XP Professional for the series of examples here, this will walk you through using Sysprep to freeze your image and prepare it for use as a base image. I am going to start out with a fully installed Windows XP Professional guest with VMware Tools installed.

1. Insert the original installation CD back into the CD-ROM drive or reconnect the ISO image you used to install the guest. Remember, the versions of Sysprep are specific to a certain version of Windows. Use the copy found on the installation CD or that downloaded specifically for your SP level.

Due to the vast number of operating systems and service pack levels for Windows-based products, I am not able to provide you with search criteria for all possible combinations. In the case of Windows XP SP2, you want the SP2 Deploy Tools for Advanced Users. Most often you can search your operating system, service pack level, and deployment tools to find the proper version for your situation.

2. No matter which way you get to it, either the `deploy.cab` file located on the original CD or via a download from Microsoft, you will be presented with a `.cab` file. A `.cab` file is nothing more than an archiving method used to distribute files. You cannot install from the file; however, if you double-click it, the file will open up and reveal its contents, as shown in Figure 4-20.

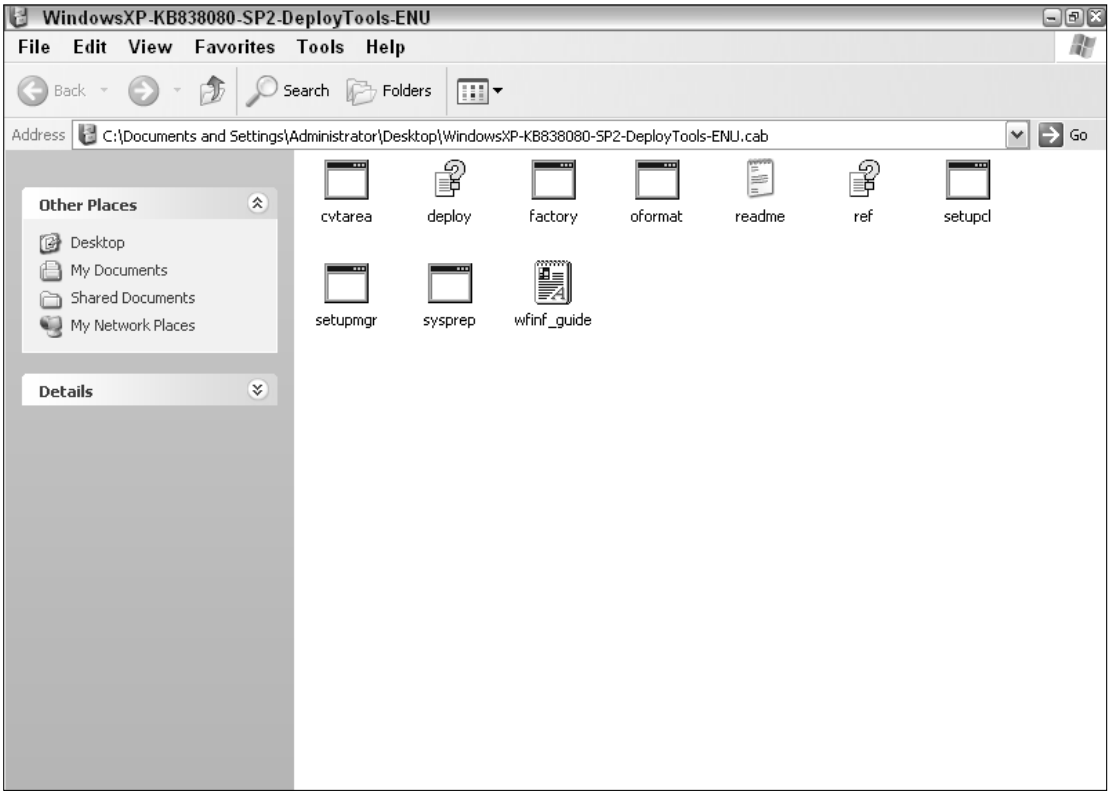

Figure 4-20

To understand what you are going to do with the files in this `cab` I should explain a little of what Sysprep is going to do for you and how. Basically, as I mentioned previously, you need each copy of your base image to be unique. To do that you are going to utilize Sysprep with an automated answer file. You will generate this answer file using one of the applications from the `cab`, `setupmgr.exe`. Once you create an answer file you want to kick off the `Sysprep.exe` file. As you will see in the following figures, Sysprep will take this answer file you create and repackage the operating system into a mini-setup state. The very next time this machine is powered on the

mini-setup will launch, the SID will be regenerated along with whatever else you specify in the answer file, and it will reboot. Once it starts back up you have a clean, uniquely identified image, and all traces of Sysprep and its answer file are gone.

3. So, how do you do this? First, you are interested in only three of the files from the deployment tools cab: `setupcl.exe`, `setupmgr.exe`, and `Sysprep.exe`. Highlight each of these files by performing a single Ctrl+Left Click, choose the Edit menu, and select Copy.

4. Next, using either the current window or a new My Computer window browse to the root of the system drive, in this example and most cases it will be C. Once there right-click in the window and select New Folder and name it Sysprep (Sysprep likes to have its files in a directory called Sysprep). Double-click your new folder and through the Edit menu again choose Paste. You should now have the three files of interest in `C:\Sysprep`, as shown in Figure 4-21.

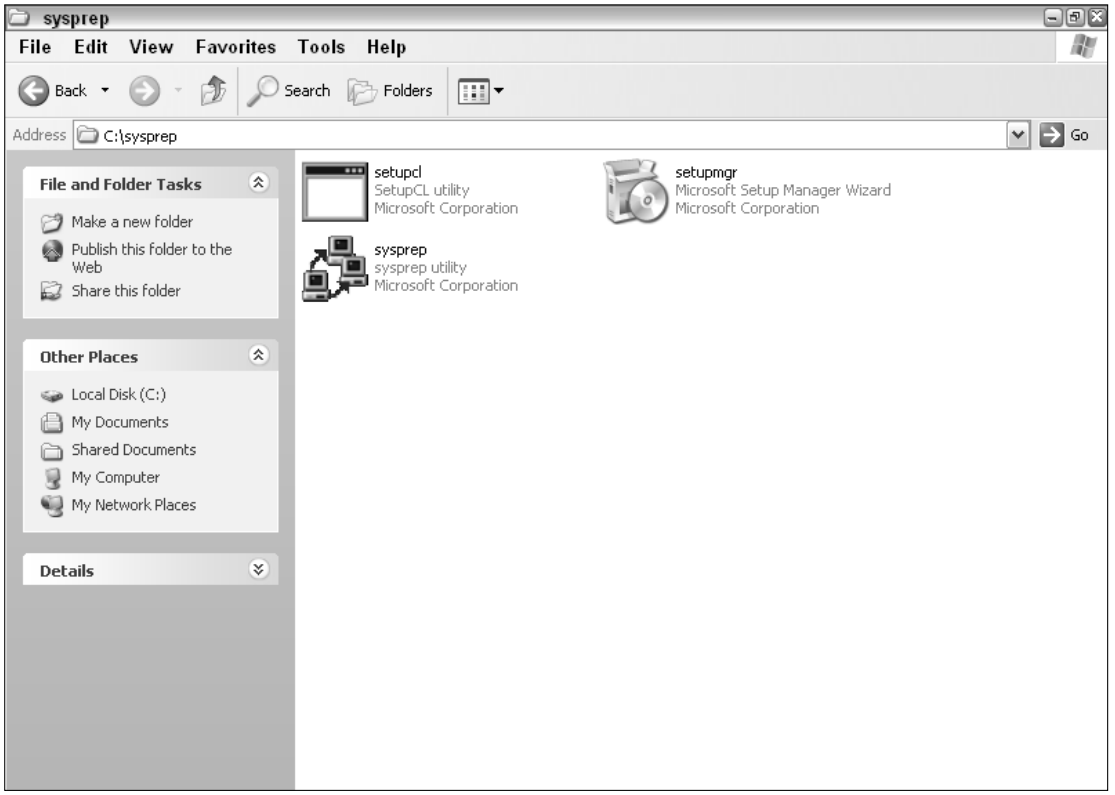

Figure 4-21

5. Now you need to create your answer file. Start the process by double-clicking the `setupmgr.exe` file shown in the preceding figure. This will produce the wizard (shown in Figure 4-22) to walk you through the process.

6. Skip the first page of the wizard because it is just a welcome screen, and proceed to the first question. You are creating a new answer file so leave new selected and click Next.

7. The Setup Manager program can be used to create a few different deployment automation files, including unattended setup. Figure 4-23 shows the options available.

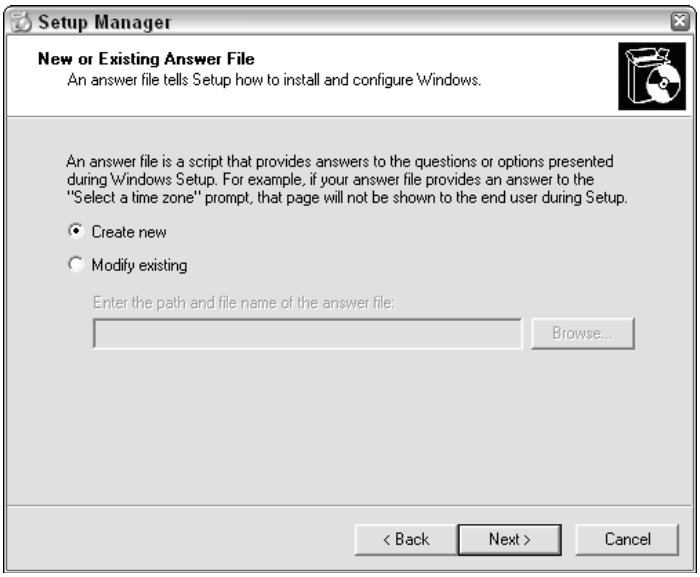
Setup Manager
New or Existing Answer File
An answer file tells Setup how to install and configure Windows.
An answer file is a script that provides answers to the questions or options presented during Windows Setup. For example, if your answer file provides an answer to the "Select a time zone" prompt, that page will not be shown to the end user during Setup.
Create new
Modify existing
Enter the path and file name of the answer file:
Browse...
< Back
Next >
Cancel

Figure 4-22

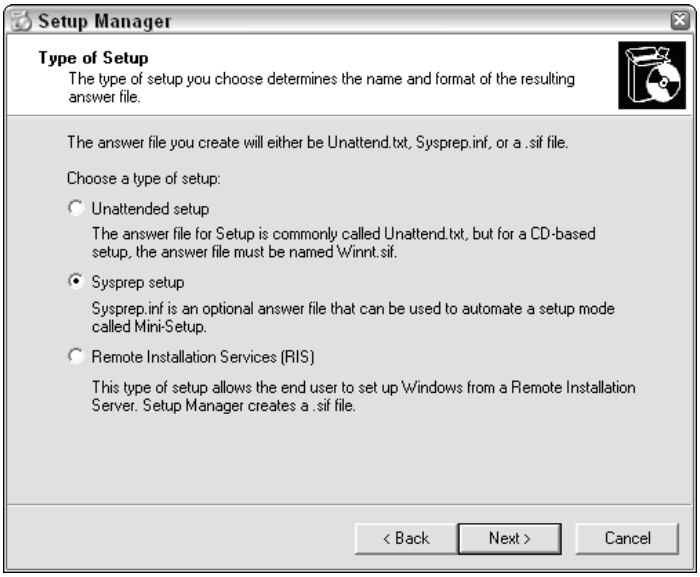
Setup Manager
Type of Setup
The type of setup you choose determines the name and format of the resulting answer file.
The answer file you create will either be Unattend.txt, Sysprep.inf, or a .sif file.
Choose a type of setup:
Unattended setup
The answer file for Setup is commonly called Unattend.txt, but for a CD-based setup, the answer file must be named Winnt.sif.
Sysprep setup
Sysprep.inf is an optional answer file that can be used to automate a setup mode called Mini-Setup.
Remote Installation Services (RIS)
This type of setup allows the end user to set up Windows from a Remote Installation Server. Setup Manager creates a .sif file.
< Back
Next >
Cancel

Figure 4-23

Because you are performing neither an unattended setup nor a remote installation, the obvious choice is Sysprep setup. If you look carefully, the Sysprep setup is used to automate a setup mode called Mini-Setup. That is exactly what you are going to do. Select Sysprep setup and click Next.

8. The next screen is simple. Just select the operating system currently installed, in the case of this example Windows XP, and click Next.

9. The next step, shown in Figure 4-24, is the final screen in the wizard before you get into the real thick of it. It is here that you need to decide whether or not you want the mini-setup to be interactive. It is possible to fully automate the process whereby no interaction is required. You unpack an image and fire it up. Sysprep will run, reboot, and leave you with a fresh machine ready to go.

   As you can see, there are two choices. I recommend you select No to not fully automate the installation. The reasoning behind this is that you will want to customize more than just creating a new SID every time you use a new copy of your base image. Most importantly you'll want each guest machine to have a unique computer name. If you automate the setup and do not allow specific items to be adjusted, you will just end up going back to each image and changing settings after the mini-setup is complete. That, of course, goes against what I'm trying to help you do and that's to create a clean base image that is just ready to go right out of the box. Select No and click Next.

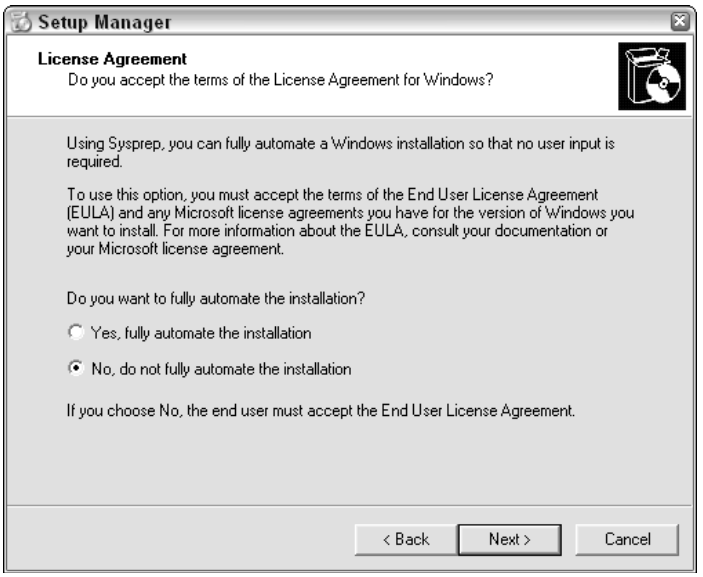
Setup Manager
License Agreement
Do you accept the terms of the License Agreement for Windows?
Using Sysprep, you can fully automate a Windows installation so that no user input is required.
To use this option, you must accept the terms of the End User License Agreement (EULA) and any Microsoft license agreements you have for the version of Windows you want to install. For more information about the EULA, consult your documentation or your Microsoft license agreement.
Do you want to fully automate the installation?
Yes, fully automate the installation
No, do not fully automate the installation
If you choose No, the end user must accept the End User License Agreement.
< Back
Next >
Cancel

Figure 4-24

This concludes the use of the wizard and places you into the main Setup Manager window (see Figure 4-25).

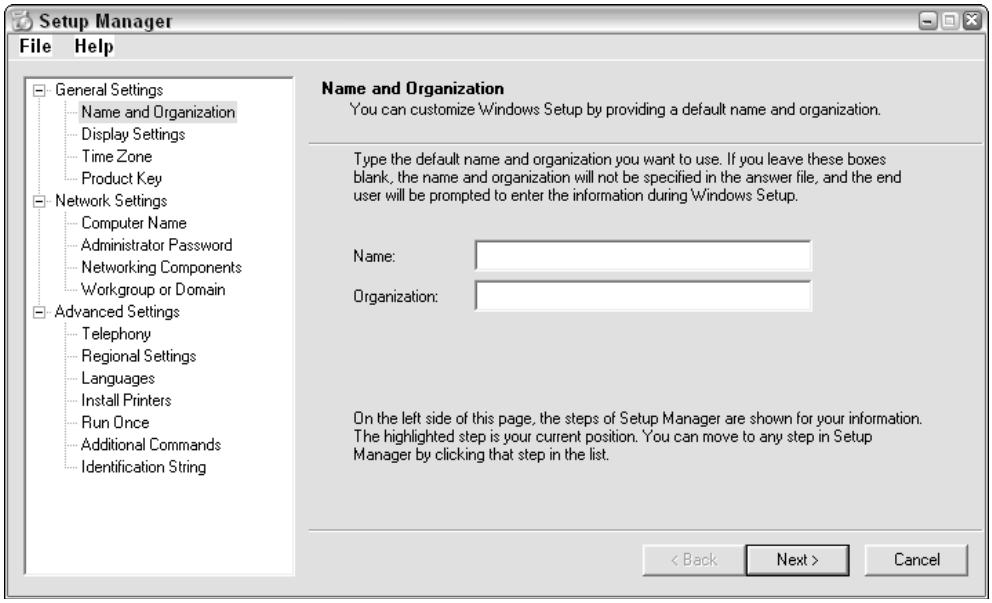
Setup Manager
File Help
General Settings
Name and Organization
Display Settings
Time Zone
Product Key
Network Settings
Computer Name
Administrator Password
Networking Components
Workgroup or Domain
Advanced Settings
Telephony
Regional Settings
Languages
Install Printers
Run Once
Additional Commands
Identification String
Name and Organization
You can customize Windows Setup by providing a default name and organization.
Type the default name and organization you want to use. If you leave these boxes blank, the name and organization will not be specified in the answer file, and the end user will be prompted to enter the information during Windows Setup.
Name:
Organization:
On the left side of this page, the steps of Setup Manager are shown for your information. The highlighted step is your current position. You can move to any step in Setup Manager by clicking that step in the list.
< Back
Next >
Cancel

Figure 4-25

I want to save you some page flipping here, so I am choosing not to show you every possible option but instead just pointing out the basics. Proceed in the main Setup Manager window as follows:

1. On the left you are given a set of categories. The Next button at the bottom of the window advances you through each one, or you can pick and choose by selecting what you want from the tree view itself. Everything here that is left blank (with the exception of items like Telephony) will prompt you during the mini-setup phase of the installation. For instance, if you leave the first screen where it asks for name and organization blank, you will be prompted every time you start up a fresh base image for a name and organization. If, however, you have a default value you want to provide for all of your images, enter it here and click Next.

   So, the more you can fill in now the less work you have to do later. I suggest you answer and fill in absolutely as much as possible except the computer name under the network section. You want the mini-setup to ask for a unique computer name every time you create a new guest from your base image. After all, that's why you are doing the interactive install. If you have a default workgroup or domain you want your test or development images to join, feel free to enter that as well. You will find Sysprep works pretty well.

2. Once you reach the last setting, Identification String, click Next and the setting wizard prompts for where to store the answer file. It should default to the `C:\Sysprep` folder, which is exactly where it needs to reside. I have to warn you that once the answer file has been saved, Sysprep does something a little odd. The main pane displays only a Cancel button. One would expect Close, but alas that is not what you get. Never fear, however; just click the Cancel button to close the setup manager interface, and the answer file is safely tucked away where it needs to be.

3. Reference Figure 4-21 or your `C:\Sysprep` folder, and the next application you need to run is called `Sysprep.exe`. This is the actual application that will repackage your image and apply the answer file. A word of caution—Sysprep automatically deletes the entire `C:\Sysprep` folder on reboot so be sure you save your answer file to a safe location for later use or else you will end up running the entire wizard again.

   Double-click `Sysprep.exe` to start the final process. Once you read and accept the warning message, you will find in the Sysprep interface shown in Figure 4-26.

   You can read through the various options here, but the only one you are interested in is Reseal. The Reseal method takes the answer file you create and along with the mini-setup option seals the operating system into a final installation state. This will all become clear once you actually see it run; for now just make sure the Use Mini-Setup option is checked and click the Reseal button.

4. After a few moments Sysprep will shut down your guest machine. Now I want you to turn right around and start it back up. This is important to verify the answer files you created in fact behave the way you intended. Regardless of whether you left any questions in the answer file blank, you will be prompted to accept the End User License Agreement. Verify all the items you expected prompts for appear, and once the mini-setup is complete it should restart your machine. At this point you should have a fresh virtual machine with a new SID and unique computer name. All the ingredients you need for a rapid deployable base image.

The Sysprep tool is not limited to use right after installation. I recommend you install the base software you want your image to contain prior to running Sysprep. For example, if you are creating base images for your development workstations, you can install Visual Studio, or the environment of your choosing, and then run Sysprep.

5. If the mini-setup did everything correctly and you are satisfied with the outcome, copy your Sysprep folder back into the root of `C:\`, start `Sysprep.exe` again, and reseal.

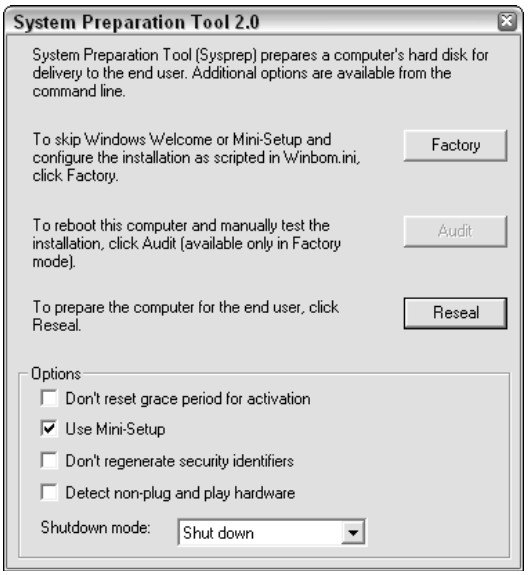
System Preparation Tool 2.0
System Preparation Tool (Sysprep) prepares a computer's hard disk for delivery to the end user. Additional options are available from the command line.
To skip Windows Welcome or Mini-Setup and configure the installation as scripted in Winbom.ini, click Factory.
Factory
To reboot this computer and manually test the installation, click Audit (available only in Factory mode).
Audit
To prepare the computer for the end user, click Reseal.
Reseal
Options
Don't reset grace period for activation
Use Mini-Setup
Don't regenerate security identifiers
Detect non-plug and play hardware
Shutdown mode:
Shut down

Figure 4-26

You now have a fully formed base image to add to your image library.

> **I should note, however, that the use of Sysprep is not required. I use Sysprep for all of my Windows-based images because the process of renaming and SID generation is all automatic. That way I never forget. You can achieve the same results by simply copying the base image files to a new location, changing the name of the virtual machine in the guest operating system, and running a SID generation tool, such as Sysinternals NewSID. This, of course, requires you to not only remember to do it, but also the manual steps required in the process. Sysprep takes care of it for you. Run it once on the base image and every single base image spawned from it will automatically take care of renaming and SID generation for you.**

Summary

Over the course of this chapter you have started to get into the use of VMware Server through the creation of your base images. These images are a crucial part of building your virtual machine library to allow you to rapidly deploy, provision, and utilize VMware Server to its fullest extent. In review, the following topics were discussed:

- What a base image is, how is it used, and what value it provides to your environment
- Creating your first virtual machine, a brief hardware overview, and look at the New Virtual Machine Wizard
- Installing your guest operating system and the possible issues that may arise
- Installing the VMware Tools package on both Windows- and Linux-based guest machines
- Freezing your Windows-based guest machines using Sysprep

Though this process may seem a little lengthy, it in no way compares to the time you will spend installing each and every virtual machine you want to create from scratch. If you take the time to assemble a well-rounded base image library utilizing the methods discussed in this chapter, not only can you take a vacation with the time you save but you gain a valuable ability to rapidly deploy new machines fully configured.

In the coming chapters the necessity of this well-formed image library becomes more apparent, especially once I start discussing how to set up your software testing environment.

In closing I want to once again stress that you need to be aware that the packaging and deployment of your base images requires you to either already possess or obtain proper licensing for the operating systems in use. Every time you copy and deploy a new version of your base image to run on VMware Server, it consumes another license. I recommend that if you are looking at a large-scale deployment with multiple images for either test, development, or both, you check into volume licensing if you are using Windows-based machines or the equivalent licensing for any other operating system you use.

5

Where Does This Fit into Your Development Environment?

Everyone's development environment is unique in its own way. That said, we all usually battle with two very real obstacles, time and money. There are a lot of steps involved in order to release software into distribution. If you take a very high-level view, there are typically four major milestones involved:

- **Development**
- **Test** (unit testing, usability testing, integration testing, etc.)
- **Release**
- **Maintenance**

All of these milestones take both time and money, usually a considerable amount of each. So how do you cut down on the time and money required, you ask? One way is with VMware Server. This chapter serves as a jumping off point for the chapters to come on how you can integrate and automate some of the steps just listed. As you will see, a great deal of time and money can be saved in development, testing, and post-release maintenance through the utilization of VMware Server.

Then, after I cover how virtual machines fit into your current development environment I wrap this chapter up by extending the discussion of base images from Chapter 4 into some coverage of creating base images for your developers' workstations.

Virtual Machines for Software Development

As I walk through various technical conferences and conventions I always like to ask the software developers I meet one question — Is your development environment inside a virtual machine? The typical reply from these developers is *no*, followed very curiously by *why*. That may be your answer as well. I want to take this section to explain why and just what there is to gain from developing in a virtual machine.

A lot of developers write software every day on their office desktop — the same desktop machine on which they perform their everyday tasks like reading e-mail and surfing the Internet. Does this make them bad developers? Of course it doesn't, but in the end it will cost you time and money, something that is already in short supply. Take a quick look at why that is.

The typical office workstation comes preinstalled with a myriad of productivity, accounting, and line of business applications. Now, by itself this may not be that bad, but when you add the multiple instant messaging applications, card games, cute screen savers, and so on that tend to clog all of our hard drives, things start to get a little hairier. On top of that you read your e-mail, which contains who knows what these days, and browse the Internet. Then you install your favorite IDE (integrated development environment) and start writing code. For the occasional developer this may be just fine, but when it comes to serious product development of code bases that will last for years to come, this is a nightmare waiting to happen.

Inevitably machines crash, software corrupts, installs fail, or people just want to erase and start over. All of these things happen, and when they do, you just have to pick up and start over. However, there is a problem. You can never get it back to the exact way it was. In terms of software development this can lead to serious issues. The third-party library utilized in the code you were working on is no longer installed; you have to go find that. Then the configuration is wrong, or you have to set up your IDE differently. You can insert any number of things that can, and usually will, go wrong. This costs time and money.

What if that were never an issue? Suppose you could create a clean development environment separate from your everyday machine that was both easy to maintain and uniform in its configuration every time. That would be a great thing to have. Well, you can have it with VMware Server.

Take a moment to ponder the points that follow. Think about how they compare with the current way in which you handle development environments within your team.

- **A common environment matching in both configuration and tools for your entire team** — Developers utilize a wide variety of tools consisting of an IDE, source control, bug tracking, project management, and so forth. Configuring them all takes time, especially if there are guidelines you enforce among your developers. An example might be source control. Suppose you want to make sure your developers are all checking in their code with comments forced via a client setting. Or perhaps you want to set up the bug-tracking client to connect to a specific server or your developers' IDE to display items a certain way. The important thing to think about here is uniformity. Uniformity is paramount when it comes to a development team. Ensuring everyone goes about the process in a specific manner with specific settings and tools is a very difficult thing to manage. Attacking this every single time is simply unnecessary, not to mention very time-consuming.

You could argue, however, that this can be accomplished by imaging the workstation. That is true, and a valid solution; however, if your team doesn't have a dedicated development machine, this process becomes tedious and time-consuming for both you and your team. Think about the time it would take to re-image a workstation and get it ready for use compared to the time it would take to turn on a new virtual machine. As you move throughout the remaining chapters in this book you will quickly learn there is no comparison to the simple, quick, and reliable use of virtual machines.

- **Quick rollout of new environments and tools** — As I mentioned previously, physical workstation images can be used for quick recovery of development environments for your team. Machine images are tied to specific hardware and as such can require an image for each workstation. Beyond that, any change in team members' hardware can render the image virtually useless, and the quick rollout of new environments becomes, once again, time-consuming. This is where the virtual machine comes to your rescue. Virtual hardware doesn't change. It remains constant for everyone using VMware Server. A virtual machine created using VMware Server will see the same hardware set regardless of what workstation it runs on. Now when your team gets an upgrade to some or all of its workstations the impact is negligible. You simply install the VMware Server Console, and they are off with exactly the same environment. If their machine crashes and they have to use another until theirs is repaired or replaced, no problem. They install the console, and the environment from the day before is right back exactly where it was left.

  An exception to this is the processor. A virtual machine in VMware Server sees the processor that is installed on the local workstation. With that in mind your users may run the same virtual machine image on different processor platforms. In most cases this makes little or no difference unless you are developing against a specific processor feature or instruction set unique to that line. One situation you may run into problems with, however, is product activation. A processor change on the physical workstation could be seen as a change in hardware by an activation scheme, like Windows XP, and force a reactivation. Because the actual processor is the one piece of virtualized hardware that actually represents its physical counterpart down to the type and model, it becomes somewhat of a special case.

- **Archive of entire development environments for major projects** — Have you ever found yourself digging back into that old code base from three or more years ago attempting to fix a bug for a customer? If you are anything like me, you will have a hard time remembering exactly what was done, how it was done, and what it was done with. Most often I would find myself searching for missing libraries so the application would compile or tweaking IDE settings to emulate the environment as it was when I wrote it. Imagine if you could just freeze everything at release so you could come back six months or a year down the road and be right where you left off. You could unplug the computer and sit it in the corner, or you could use a virtual machine. As you learned in Chapter 4 a virtual machine is nothing more than a set of files on the server. As such they can simply be turned off, compressed, and stored. No wasted hardware and no struggling to make things work a year later. You simply retrieve the files and turn it on. You have ready access to the exact environment you left when you finished the project. The hours this saves alone is reason enough to develop within a virtual machine.

  This is in no way a replacement for good documentation. However, imagine if your documentation could state that the code was written on a certain image, its name, and where it is located. The strength of your documentation is now double because you are not only stating specifics about the development, code base, and process, but also providing the exact environment in which it was written.

Now that I've familiarized you with some of the benefits of using virtual machines during your development process, you can next see how they become an indispensable tool during your testing phase.

Virtual Machines for Software Testing

Testing takes on many different forms, too many to list here. This section touches upon only one facet of software testing, platform testing. The reason I choose to hit platform testing over the other forms of software testing is simply due to time and money. Stress testing, load testing, vulnerability analysis, and so on often take little in terms of resources like hardware, although they are rather time-intensive. Platform testing is often the most emphasized and daunting step in the testing process.

As you develop software for commercial use you have to take into account the customer's environment. Believe it or not there are still quite a few of your customers that run a less than up-to-date operating system on their computer. Perhaps you're developing an application that runs on both Windows and Linux. That being said, you may choose, or be required, to test your application on a variety of different platforms and operating systems. In a world without virtualization you would either have a bank of test machines that cover the range of platforms and operating systems you support or simply have one or two machines that you constantly rebuild time and time again with different operating systems throughout your testing process. I don't think I even have to point out the cost involved in a test bank nor the time involved in re-imaging a machine to switch testing environments. This is where VMware Server shines.

So how exactly can virtual machines make this process easier? Virtualization as you know allows you to house many different "virtual" machines on one physical server. Say, for example, you have a single VMware Server. On that server you create your bank of various platforms to test your application against. For instance, you might want to test your installation against an RTM (Release To Manufacture) version of Windows XP as well as a version with Service Pack 1 and again a version with Service Pack 2 installed. Do you create one machine with a clean RTM version of Windows XP installed, test, then install Service Pack 1 and repeat? No, absolutely not. Create yourself three separate virtual machines, each with the appropriate patch level you want to test. These are called your *base images*. A base image is a static virtual machine you create that serves as a jumping off point for all other virtual machines you want to utilize in the future. By taking the time to create several good static base images you will find yourself always having just the right install available to you when you need it.

And you can further save yourself some work here. Create your first virtual machine as a Windows XP RTM and then copy that virtual machine twice and add the additional service packs and updates from there. In the end you've installed Windows XP only once, and you have three base images at the appropriate service pack levels.

> *RTM or Release to Manufacture refers to the original release of a software product. In the preceding paragraph the RTM version of Windows XP refers to the original version released by Microsoft at the product launch.*

Platform testing requires a degree of uniformity. If you test your application on a specific version and patch level of an operating system and it fails, the ability to test that same install again, after you correct the bug, with exactly the same scenario, is paramount. To help assist with this VMware Server offers a couple of different solutions: snapshots and nonpersistent disks.

I certainly don't want to leave Linux platform testing out of the mix here because it can probably benefit the most from this. With the number of Linux distributions increasing every day, the need to test certain Linux-based applications across several different distributions becomes even more necessary. Take, for instance, a KDE targeted notepad knockoff you develop. You write the application and test it against what you consider to be the three distributions you want to target, SUSE, Red Hat, and Ubuntu. If you haven't used Linux before, I'm here to tell you that every single distribution packages different libraries and applications for the default installation. Just because package *x* that your notepad knockoff requires is installed by default on Red Hat doesn't mean it will be on SUSE or Ubuntu. The flexibility of VMware Server within platform testing shines here because now you can easily identify the discrepancies present due to the different distributions and take steps to solve then. Essentially this is no different than the goal of platform testing on Windows-based systems. The significance lies in the fact that there are a greater number of differences between a base Red Hat install and a base SUSE install as opposed to those between a base Windows XP Professional and a Windows Server 2003, for example. The efficiency of your Linux-based platform testing can increase significantly through the use of VMware Server.

Platform Testing with Snapshots

A snapshot is simply what it sounds like. When you take a photograph of the family on vacation, the exact moment is captured and while events continue to take place after the photograph was taken, you can still look back and remember that moment. If you create a virtual machine with Windows XP installed and create a snapshot of it, you freeze the disk exactly where you left off after the install. This saves all subsequent changes into a new virtual disk, or snapshot, file. You can then, via the console or API, revert to a previous version of the virtual machine.

Say you have an application that relies on a framework to be installed, the Java runtime, for example. You can fire up a copy of your virtual machine base image and immediately create a snapshot. This freezes the copy where the base image left off. Now install the frameworks needed and create another snapshot. Then install and test your application. Now, not only do you have a snapshot to revert to before you installed your application to test with the framework already available, but you also have a snapshot to revert to if you want to test, for instance, a bootstrapper that automatically installs the framework for you. All of this can be accomplished with a few clicks as opposed to formatting and reinstalling an image on a physical machine. Snapshots can be taken of your virtual machine by selecting the snapshot icon from the toolbar or from the VM menu in the menu bar with the machine's tab selected in the main window.

If you revert to a snapshot, all changes made after the snapshot was taken are lost. In the preceding example if you revert to the snapshot taken prior to the framework installation, all changes made after the first snapshot are lost, including the snapshot taken after the framework was installed.

Platform Testing with Persistent and Nonpersistent Disks

Persistent and nonpersistent disks require a bit more explaining. Both persistent and nonpersistent disks fall into a category VMware likes to call *independent disks*. An independent disk is a flag set by you that indicates whether you want the contents of that virtual disk to be included in snapshots taken of that virtual machine. It for all intents and purposes isolates that disk from the snapshot process.

For example, say you have a virtual machine with two disks — the primary is where your operating system is installed and the secondary is simply a data drive. This could be compared to having two hard disks in your physical computer where the second hard disk stores documents, photos, music, and so on that is independent of the operating system entirely. Now, say you decide to create a snapshot of the virtual machine. The snapshot essentially captures the state of the operating system, or more accurately the state of all non-independent disks, at that point in time. By default all virtual disks are not independent, which means both will be captured in the snapshot. Do you want your data drive to be captured? Possibly not, because it may contain data that you want to persist throughout all forms and iterations of the virtual machine's state. If you open the virtual machine's setting and choose the secondary drive you can, through its advanced settings, make that disk independent. This will *isolate*, if you will, that secondary drive from the snapshot process. Now, go back and make a snapshot of the virtual machine. If you fire up and use this virtual machine and save new data to the secondary drive, that data will remain even if you revert to the snapshot taken beforehand. This is all because the disk has been marked independent.

As you may have noticed under the advanced settings for your virtual disks, independent disks have two basic modes, persistent and nonpersistent:

- A **persistent** disk is one that writes its data directly to the virtual disk file immediately and permanently. Data written to a persistent disk remains even when the virtual machine is reverted to a snapshot. This is the default behavior of all independent disks.
- The second option, **nonpersistent**, you will use only in the most special of circumstances. A nonpersistent disk is one that does not write any data to the drive. A nonpersistent disk is essentially read-only in nature. You can interact with the drive and even save data to it during the normal operation of your virtual machine; however, once you power off that machine all changes to that nonpersistent disk are lost.

  I can make that a little more concrete with an example. Managing IT for a company requires you to evaluate and test a lot of software applications. Somebody is always wanting to install product A or buy software B, and this usually requires IT to evaluate and test the package prior to installing it on a workstation on the network. This can easily be accomplished using a virtual machine and independent, nonpersistent disks. Copy over a base image for the operating system that is predominant on your network and fire it up. You must complete the Sysprep steps and make sure the system is configured correctly prior to setting it as nonpersistent. Once you complete that setup, shut down the guest and set the virtual disk as nonpersistent. Now, on every boot, the virtual machine starts as a clean sandbox to test these random pieces of software. Load it up, test it, and shut it down. All traces of it are gone because that is the very purpose of the nonpersistent disk, to give you a clean slate on every boot. Really it is a seldom-used feature, but as you can see, it does have its place.

For more detail about setting up your virtual machine to use nonpersistent disks, check out Chapter 6.

Virtual Machines for Post-Release Maintenance

So you have made it through the development and testing process for your application and released it to your customers. Now comes the bug fixes or possibly maintenance releases for your application. Depending on the longevity of your application or the nature in which your customers utilize it, you could possibly be supporting the application for many years to come. Over time this becomes increasingly more difficult as developers rotate in and out of your group as well as changes in technology. VMware continues to support virtual machines from previous versions so you can ensure that your images, test platforms, and development environments remain available. The usefulness of VMware Server in this stage of the development process can be outlined by the following two points:

- **Archive of the development team's virtual machines used during the development phase** — As I discussed previously, moving your development environment into virtual machines offers a multitude of advantages over conventional machines. One additional advantage, and one I consider to be the most important, is the ability to archive the developer's environment for later use. A good habit to get into is using a development image for each major project. Once the project is complete and pushed to release, you can archive the image to disk or DVD to allow for later reference when the need arises. This allows the developer assigned to post-release bug fixes or maintenance to quickly locate and power on the exact environment he or she, or another member of the team, used during product development. This offers a huge increase in productivity and turnaround time. By the developer not having to adjust his or her current environment to handle the previous project, you also run less of a risk of the current project suffering from setbacks due to mishandled code and configuration changes, in addition to saving the time it takes that developer to adjust back to the current project.
- **Rapid platform availability for quick support tasks and bug investigation** — The rapid availability of platforms for support analysis is essential. For instance, say a customer contacts you for support on one of your products. Right away you can load up a testing image in VMware Server and install the product either to familiarize yourself with the issue or attempt to re-create it. Having a solid set of testing images also allows you to re-create environments that more closely match that of the customer. For instance, suppose the issue cannot be re-created on your testing image running Windows XP; however, you know the customer is using Windows 98. Without virtualization you would need an additional machine with Windows 98 or you would need to wait until you could install it on a free system before you could test. With virtualization, you simply fire up your Windows 98 image and test away. This shortens turnaround time for the customer's issue and increases the ticket closure rate for your help desk.

Creating Base Images for Your Development Environment

This next section extends the discussion from Chapter 4 on base images. I want to take a few moments to talk about base images for your developers' machines. As I outlined in the preceding sections of this chapter, there are numerous advantages to running your development environment within a virtual

machine, and I highly advise you do not overlook the benefits. With that being said I want to take a few moments to share some of my past experiences with getting virtual development environments off the ground and accepted by my teams. The following is a series of steps I follow when creating new virtual development environments for my team:

1. Plan
2. Collaborate
3. Create
4. Test
5. Deploy

The sections that follow discuss each of these steps in turn.

> **For the Linux developers out there the five steps I list apply as well. Though I make use of Microsoft-based references like Visual Studio 2005 and .NET, the very same applies to Apache, Perl, Eclipse, and so on. I have used these five steps to create viable effective development images in both Linux and Windows.**

Planning

Everything requires planning. This might sound simple enough, but a clearly defined environment is crucial in order to get your development team to adopt this method of writing code. Start by identifying the requirements for the project at hand. If, for instance, the new project is based on the .NET Framework, you need to identify the specific requirements of that project. Is it web-based, or does it require the development of web services? Will the development team host a development copy of the web site or web service locally or on a joint virtual machine with Windows Server 2003 installed?

I also encourage you to think about virtual networks. Do you want your development machine running on the same subnet as your regular machines? Maybe and maybe not. One thing to keep in mind while thinking about this is that a lot of Microsoft applications now like to "reach out to the mothership" for updates and even for help files. This is especially true for Visual Studio 2005. If you want your developers to have this opportunity, then put some careful consideration into your networking configuration now to avoid heartache later. There are some definite advantages to placing your development machines in a different subnet — one of which is keeping your development work secure. Although this will not solve all your problems in this area, it is certainly a start.

An example to help define the process may help clarify this. Suppose you were just assigned a project involving a new ASP.NET site. Your team, which is made up of a few developers, will create and test the site internally before pushing it out to a remotely hosted production environment. This being a web-based project brings you to your first major decision. What version of the .NET Framework are you developing in and what is your IDE of choice? This is a very important question to answer to determine the direction of your development image. First and foremost for ASP.NET development your developers are going to need access to a web server capable of handling ASP.NET content. If you still develop code in .NET version 1.1 and use Visual Studio 2003, you will have to plan on either installing IIS onto a Windows XP image for them, provide them all with Windows Server 2003 images with IIS 6 installed, or create a separate web server they all can share. If you develop the site using .NET version 2.0 and Visual

Studio 2005, they already have everything they need because Visual Studio 2005 provides its own internal web server for ASP.NET development. Confusing? That's why I stress planning. The choice you make here will make or break your deployment.

Say for the sake of this example, and your sanity, you are writing this site in .NET version 2.0 with Visual Studio 2005. That being the case you decide on Windows XP Professional for the base operating system of the development image. Because you have your trusty base images already created, copy over a Windows XP base image and rename it to your development image. As I described in Chapter 4, I typically name base images as BASE followed by the name of the operating system. With development images I use the same format; however, I replace BASE with DEV. Feel free to choose a method that best suits your environment.

So, you know your base operating system is going to be Windows XP Professional. What's next? Now you want to sit down and list out every piece of software used by your development team. This can range from the IDE to bug trackers and project management tools like Microsoft Project. The important thing here is to list out everything you want your development team to utilize during the process. Trust me, if you leave something out and have to go around installing items after you deploy the image, your developers are going to be less likely to adopt. Once you have everything listed that you need to install, it's time to move into the next step, collaborate.

Collaborating

Round up your project managers or, depending on the size of your team, round everyone up. Make them a part of the planning and that will not only ensure a good clean development image gets created, but will also help you during the deployment phase. I recommend you forward around the list of development software to your team with a focus toward soliciting their input. More than likely you will find some piece of software was overlooked.

Once that is complete you can move into probably the most painful part of the process, enforcing standards. Because you will be creating their development environments for them, this is a perfect opportunity to force upon them certain coding standards through the use of settings in the IDE. There is a world of difference between asking someone to change a setting and changing it yourself. For example, if your team is C# focused, you may want to set curly braces to be on a new line or perhaps spaces before and after parameters. Maybe you want to enforce certain source control best practices that can be achieved through configuration of the local source control client. Regardless, you can set up all these and more through careful planning and collaboration.

Creating

Ah, here's the fun part. Now you get to take all the feedback, input, and planning from the preceding steps and start creating your development image. If this step takes a while, then you are doing it right. Time will go into this image, and it should because you will be asking your development team to live inside this image for the duration of the project, which could be months or longer.

Start with a base image, for this example Windows XP Professional. Right off you want to install any and all updates you can get your hands on for the operating system. You should treat your development image just as you would any physical machine on your network when it comes to updates. This ensures you have a good, up-to-date operating system for your development machine. But there's a catch. Updates that are not distributed uniformly after the image has been deployed break one of your main goals, uniformity. Be careful with updates after deployment for this very reason. I'm not saying they are

less important. They just require a higher degree of monitoring to ensure all images receive the same patches in order to maintain your uniformity. This is one of the main reasons I stress installing all the updates now, while the image is still being created. Continue to work down your list of software, installing each one by one.

Wait — is antivirus software on your list? It should be. Just because these are virtual machines doesn't mean they are immune. Make sure you run the latest antivirus software on all your development machines. Be just as diligent here as you would with your regular desktop machine. For some reason, like updates, people do not believe this to be necessary. Don't be fooled by that notion.

Now, a couple of things to think about while you are doing this:

- First, remember this is a base development image. If you fill up the recycle bin with files, empty it before you deploy the image. Everything you leave behind, your web history, cookies, document history, and so on, will be on each and every developer's machine. Clean up after yourself and things will go a lot smoother.
- Second, start your programs at least once. Many things like Microsoft Office and Visual Studio do not create certain items until they are run at least once. Of course you will be customizing the settings in things like Visual Studio to conform with your software development guidelines, so this shouldn't be a problem.

Now I want to talk for a minute about the use of snapshots. As you may remember, snapshots capture the state of the virtual machine at a specific time. You might think you need to create a snapshot right out of the box so you can go back to the base image prior to any software installation. You also might want to create snapshots after certain milestones like installing the IDE. You can, of course, handle this however you want, but I recommend against it. Snapshots, though very convenient, are simply an unnecessary complication in this scenario. They increase the disk space the virtual machine occupies, making deployment more difficult. Stacking snapshots several levels deep, while technically will work just fine, adds some serious bulk and complication to your image. Ask yourself if you are ever going to need to roll that virtual machine back. If the answer is no, or even probably not, then you most likely don't need a snapshot at all. It is one of those cases where just because you have a tool doesn't mean you always have to use it.

Finally, the last step in creating your base development image is Sysprep. Just like the base images you created during Chapter 4, each development image you create will need unique SIDs and names on the network. If the details are a little fuzzy refer to the Sysprep steps outlined in Chapter 4. The same procedure applies regardless of whether it is a base image or a base development image.

> *As I pointed out in Chapter 4, if you do not use Sysprep for whatever reason you still need to ensure that your development images all have unique SIDs as you deploy. This is the primary reason I use Sysprep — it handles this automatically on first boot. Other tools like Sysinternals newSID require you to boot the system, run the tool, and then reboot; a manual process that violates our goal of time efficiency.*

Testing

You should already be quite accustomed to testing. Before you test your image I recommend you make a copy of it. This will save you time later when and if you decide the current image is ready for deployment because you will not have to Sysprep and clean up after yourself again.

Some things you want to look for are the operation of Sysprep and if it provides the prompts you configured for user interaction without error. Also, fire up each and every piece of software you installed in the preceding step to ensure everything is working as expected. Check your trail; make sure you cleaned up after yourself. And lastly, sit a developer down in front of it and ask him or her to work. Give any developers a few hours, and they will find the problems with your setup. You will save yourself a headache if you do this now rather than later. Trust me when I say the first iteration will probably not be the last. Setting up a deployable development environment with the proper tools and configuration is rather difficult in the beginning. I promise, however, that it will become much easier, especially if it is an environment that can be used for multiple projects.

Deploying

So you've carefully planned, collaborated, created, and tested your development image. Everything works perfect, right? Absolutely. Now comes the time to push this out to your development team. There are basically two ways to accomplish this: hosting each developer's image on a VMware Server or having each developer run the image locally with VMware Player.

You're probably saying, "What is VMware Player?" Well, *VMware Player* is simply a lightweight virtual machine console offered for free from VMware that allows you to run any preconfigured virtual machine effortlessly. VMware Player does not offer the bells and whistles of the server console or VMware Workstation; however, it is more than capable as a simple free tool to utilize your preconfigured virtual machines. To be completely honest, I tend to run local virtual machines in VMware Player regardless of whether I have VMware Workstation installed. VMware Player is not your choice, however, if the end user needs to be able to modify the virtual machine's configuration or the local workstation just doesn't have enough memory to support running the virtual machine. Another nice feature of VMware Player is its ability to open and run Microsoft Virtual PC and Microsoft Virtual Server virtual machines. You can't beat that for flexibility. VMware Player can be installed right alongside the VMware Server Console or VMware Workstation without issue.

The method you choose will most likely depend upon your budget and team size. Hosting the development environment on a VMware Server gives you the opportunity to control their environment as well as save costs when it comes to purchasing developer workstations. If you house the image on a server, your development team really doesn't need that fancy 64-bit machine you and I both know they want because the main load of running that machine stays with the server. Of course, because this is a VMware Server book, I'm obviously going to lean in that direction; however, if your team is large or you simply can't afford a server capable of handling that kind of load, then VMware Player is your choice. You will still need VMware Server or VMware Workstation to create the images for use in VMware Player.

Choose your poison and deploy your image. Don't expect this to be an easy process. Creating and deploying your development image is not the battle. Getting your development team to see the benefits of working inside a virtual machine is. It may very well take an entire project cycle to get everyone on board, but once they start bug fixes or a maintenance cycle and all that is involved is restoring their old image from that project to get up to speed, they will sing your praises.

Now that you've successfully deployed your first development image you can say goodbye to a lot of the pains from years gone by. Ever had to walk through your team to find out who has the CD for software *x*? No more; now you control the development software. Productivity, uniformity, and cost reduction. My hope is that you clearly see the benefits of software development within virtual machines. If not I encourage you to give it a try anyway. I trust you will be amazed nevertheless.

Summary

The goal of this chapter was to whet your appetite for what VMware Server can offer you and your development team as well as to prepare you for creating your first development image. In review VMware Server can offer the following benefits to your developers and your testing scenarios:

- **Development within a virtual machine** offers you the ability to maintain clean, uniform development environments that will increase efficiency, reduce hardware costs, and reduce downtime.
- **Software testing with virtual machines** offers you virtually unlimited possibilities when it comes to testing various platforms. The ability to not only consolidate your test bank into one physical machine but to allow for quick test scenario development through the use of base images is a crucial element to reducing your costs and increasing productivity.
- **Post-release maintenance with virtual machines** allows you to retrieve the virtual machines, archived at release, on which the application was developed. This saves time by allowing a developer working in a post-release scenario to gain a quicker grasp on the issue and provide a quicker turnaround time for the customer.

Throughout the discussion of those three scenarios I pointed out the benefits of software development from within a virtual environment.

The final section of this chapter walked you through the process I use to create virtual images for my development team. Remember the steps are as follows:

1. **Plan** your development environment by analyzing your current project and the needs of your team.
2. **Collaborate** with your development team to ensure your development image will withstand the rigors of everyday use during the product cycle.
3. **Create** the virtual machine utilizing one of your base images and the list of software assembled during the previous two steps.
4. **Test** your virtual image to ensure that all the software functions correctly.
5. **Deploy** the development image to your team.

If careful consideration is given to the proper setup and configuration of your development environment prior to deployment, I assure you the development team will see the benefits and jump onboard. The productivity and cost-saving benefits are well worth the effort.

The next chapter starts to bring everything together. I discuss organizing your test and development library of base images as well as a library of ISO images of software you use regularly within your virtual machines. Then, looking even further ahead, I dive into the VMware programming interface and start to discuss how you can integrate VMware Server into your testing and Quality Assurance (QA) environment.

6

Bringing It All Together

The last couple chapters have taken you through base images, development images, and ways to set up your development environment. Now that you have that background, I want to take a look at some finer overall details that I couldn't go into in the previous chapters. This chapter brings everything together from the past two chapters with a few added best practices in my opinion.

Although you create base images to shorten the actual time and labor involved in creating a new virtual machine, if the base images are managed improperly, you can easily find yourself putting just as much work into your images as you would a fresh install. Over the course of this chapter I talk about some ways in which you can keep a firm and productive grasp on your image library.

Organizing Your Image Library

By now you have no doubt thought about what images you need for your environment. Whether your library contains just a couple of images or hundreds, the time you put into your library now will pay off tenfold after the first few weeks of use.

One of the more common mistakes made when assembling an image library is lack of organization. Many believe simply creating the images is good enough. But because you are dealing with massive archives of sometimes 5 or 10GB in size, making sure you keep good accurate records of what an image contains and its intended purpose is paramount.

Proper Image Naming

Seems simple enough doesn't it? Honestly, this one can get you. Imagine if you named all of your Windows XP virtual machines simply Windows XP. Questions you will no doubt ask yourself later down the road are, "What service pack level is on that virtual machine?", "Does it have this or that already installed?", and so on. At some point you will find yourself loading virtual machine images just to see what exactly they are. It is almost inevitable. There are, however, some simple ways to avoid this waste of time, and one is through proper image naming.

When referring to the naming of your virtual machine images I am not only talking about the image directory itself, but to the entire virtual machine. From the name you give a virtual machine when you create it to the name you use when saving the virtual machine configuration file at the end of the machine wizard, it should be consistent and meaningful. If you need to rename any previously created virtual machines you can do so via the server console virtual machine settings.

As I alluded to back when you were creating your first image, I have adopted a naming scheme over the years that works for me quite well. Stop and think for a minute about what exactly you would like to know about an image at first glance. I'm not talking about information about your image from within the VMware Server Console; I'm talking about just files stuffed away somewhere in a directory for storage. Your base images should not live in your console inventory anyway; they are pristine templates of virtual machines that you want to copy and use for a specific purpose. The best thing to do is store them away somewhere, possibly compressed, for later retrieval. It's during that retrieval stage that you will want to be able to take a quick glance and know exactly what a specific image is for. Because you've had time to think about the preceding question, I want to list for you what I like to know about an image strictly from its name:

- Type of image
- Operating system
- Patch level of the operating system

These are three key pieces of information for you to quickly determine what image you need to grab. Once you have identified what information you want in the name, you determine how to format that information. While you are thinking about how to format the names I want to mention that later you will be scripting the retrieval, startup, and shutdown of these machines as part of the software testing phase. This means you should put some careful consideration into the name to be able to parse it via some sort of script or code later for automatic retrieval. I propose the following scheme that I have used for quite some time. It works well and is the format I use throughout the rest of the book:

```
<type>-<operating system>-<patch level>
```

To make this a little clearer consider the name for a virtual machine that is a base image, runs Windows XP Professional, and has Service Pack 2 installed:

```
Base-WinXPPro-SP2
```

First is the type. You may have several different types of images based on your environment but I basically have three different types of images that I'll be discussing: Base, Dev, and Test.

- A **Base** image is exactly what you created in Chapter 4. It's an image that you base all other images off of. It should never be used itself and always be copied and renamed before basing any other image or virtual machine off of it.
- A **development**, or **Dev**, image is what refers to the images you created in Chapter 5 for your development environment. These typically contain added software packages on top of what the base image provided to facilitate software development.
- Lastly is the **Test** image. This image will most likely be the one you are addressing from within your scripts because automating your testing is a primary goal of this book. Testing images will most likely have different hardware configurations, perhaps configured with an independent, non-persistent disk, for example.

Next is the operating system name, and you will notice that I abbreviated Windows XP Professional because, honestly, who wants to type all of that. Seriously though, if you choose to abbreviate the operating system or any other part of the image name, please make them consistent throughout your library. I have listed some examples of operating system abbreviations I use at the moment and their meaning to give you a better idea.

- WinXPPro — Windows XP Professional
- WinXPHome — Windows XP Home
- Win2kServer — Windows 2000 Server
- Win2k3Std — Windows Server 2003 Standard
- RHELES — Red Hat Enterprise Linux ES Server

I could go on but the point has already been made. The choice is really up to you and your host server's operating system limitations on file name length. VMware Server allows for virtual machine names well beyond what you should ever use.

Last is the patch level. This is an important tag because if you are doing any kind of serious software testing you need to have multiple copies of an operating system at various patch levels. For example, just prior to the release of Windows XP Service Pack 2 there were problems with various pieces of software with regard to the new security implementations Microsoft put into that service pack. The ability to test on not only Service Pack 1 machines but also on Service Pack 2 machines was crucial to developers who had the unfortunate experience of having problems with that service pack. This may not even be something you want to track for your purposes, but I highly recommend you give it some consideration. If you need help imagining a scenario, consider the customer that calls because your application is crashing on Windows XP, but it doesn't on yours. You eventually find out they are running SP1, and you are testing on SP2. Stranger things have happened. One final note about patch level, and then I'll let it go. Don't forget about original release versions, or RTM, as I've mentioned before. If your image does not have a service pack installed, just tag it as RTM.

Before I wrap up proper image naming I want to stress that this is a highly flexible idea. You can pack as much or as little information into these names as you want. I once ran into a fellow developer who wrote software that interfaced with Microsoft Exchange Server. His frustration was that with every Exchange Server service pack that was released his software would break (unfortunately just par for the course when writing against Exchange). Nevertheless, I took the opportunity to introduce him to VMware and explained how he could end the battle between formatting, reinstalling, patching, and repeating ad nauseam. Of course this means creating testing images of not only specific operating system levels but also certain server-based applications and their respective patch levels. Consider the topic at hand and think about how you would name your images based on that criteria. One possible solution could be the following:

```
<type>-<operating system>-<patch level>-<server app>-<patch level>
```

Or to follow the example utilizing Exchange Server 2003 SP2:

```
Test-Win2k3Std-SP1-Exch2k3-SP2
```

Of course that's just an example, but the point I want to illustrate is that this scheme can be anything you want. Just keep in mind consistency and the ease of being able to parse that information at a later time through code.

> *Keep in mind that if you want to make the network name of the virtual machine's operating system match that of the virtual machine itself, you are limited to 15 characters on Windows-based systems. For that reason the notes field that VMware Server keeps for every virtual machine may be useful to you. If you open a virtual machine in the console from your inventory and select its tab from the main pane, you'll notice a free-form notes field in the lower left-hand corner. This allows you to provide specific notes about the virtual machine, its setup, purpose, and about anything else. This comes in handy for the machines where the configuration is rather complex for a special name. Bear in mind, however, that you cannot parse the notes field from within a script so do not use it as your sole way of marking the images in your library.*

Archiving Your Images

Even by today's standards these images can grow rather large. With that you may decide to compress the base images into a more compact structure in order to facilitate easier storage on disk or perhaps on DVD. A base Windows XP Professional with SP2 image takes up approximately 2.3GB of space on disk but can be compressed by around 40 percent, which brings it just below 1GB in size. That is a considerable savings to say the least. Aside from the obvious advantage of saving space also comes the ability to neatly store your image in one file as opposed to having to manipulate an entire directory.

If this interests you, I suggest WinRAR. The RAR format compacts the virtual machine rather well and allows you to fit an executable extractor to the archive. This will allow for easy decompression without having WinRAR installed and just increases the ease of use overall. For Linux users nothing really beats a tar with GZip. There may be better compression available, but the availability of those two packages on almost every Linux distribution makes them an obvious choice. There are of course many different archiving programs available. Feel free to use whatever one you are most comfortable with.

Keep in mind this is to help with disk space issues as well as overall organization. Though you gain organization and disk space by archiving your virtual machines, you can quickly lose that advantage in the time it takes to restore an archive for use. It takes quite a while to extract 2GB worth of data from an archive, regardless of the format. Some could argue that it is much faster to copy 2.3GB locally than it is to extract it from a 950MB archive, and they would most likely be correct. However, archiving your images offers some ease to the organization because the entire virtual machine directory is stored within a single archive file, allowing you to store multiple images in a single directory as opposed to multiple directories within a directory.

Preparing Base Images for Use

So I have talked about base images and images for production use, such as the development image, but there is some ground in between that hasn't been covered. How do you get from the base image to a full-blown production image? There is not any real rocket science involved; however, there are a few steps you should take in order to maintain the well-organized environment you have no doubt started to create.

You have taken great care up to this point in giving your base images specific names that follow either the proposed naming scheme discussed previously or one you have created specific to your environment. True to form your production images will need their own naming scheme in order to keep the

chaos to a minimum. A couple of ideas have already been discussed in this chapter, namely the Dev and Test prefixes mentioned previously for production images. These, however, originated from a Base image in your library and as such carry the name of that base image until you change it. So the pressing question is, "What do you need to change?"

When you create a new virtual machine with VMware Server and provide it with a specific name, only two things actually get that name, the directory the virtual machine is stored in and the display name within the VMware Server Console. Everything else, the virtual disk files, the configuration file, and so on, still carry the name of the target operating system you choose during the New Virtual Machine Wizard. If you take the Custom path in the wizard, you can also rename your virtual disk file. This is not as important as the directory and virtual machine name; however, it does add another level of organization to the process. The only thing that matters is that the machine is somehow uniquely identified by your naming scheme.

> *One problem exists with renaming the virtual disk file. When you copy over a base image for use, one of the first steps you take is to rename it into the role it will fill (test, dev, and so on). The virtual disk file is a little more difficult to rename, and therefore, you might want to consider leaving it at the default. In order to rename your virtual disk files you must rename the file(s) and edit the configuration file by hand to reflect the naming change. You can also delete the virtual hard disk from the virtual machine settings under the console, read a new virtual disk, and point it to the renamed file.*

Consider this example. Say your VMware Server directory for virtual machines is `C:\Virutal Machines`, and you have created a beautiful base machine called `base-winxppro-sp2`. This base image is located in `C:\Virutal Machines\base-winxppro-sp2`. Now, it's not much of a "base" image if you just leave it lying around in the virtual machine directory. You want to protect your base images by separating them from the rest of the pack. Skipping the archiving discussion for now, you can copy the entire `base-winxppro-sp2` directory into your base machine directory that you just created at `C:\Base Images`. Now, you can make the image read-only, you can archive it up, zip it, or whatever you want, but the most important thing is that this directory is out of reach of VMware Server and the images you've created will remain in pristine condition.

Now, suppose you want to create a new virtual machine based off the base image you just tucked away. First you want to copy, and I mean just *copy*, the base image directory, or extract the archive file if you compressed the image, back into the production directory at `C:\Virutal Machines`. Next, you want to change the name of the directory you just copied in. Say in this example you are creating a testing image. Rename the directory to `test-winxppro-sp2`. This will at the very least keep your directory structure unique, but you aren't quite finished.

Next is the display name for the virtual machine. This name is used only to identify the virtual machine from within the VMware Server Console; however, it is still important.

Open a new VMware Server Console window and choose Open Existing Virtual Machine from the main pane. This will present the dialog shown in Figure 6-1.

Select Browse and locate your virtual machine's configuration file. This should be located under `C:\Virtual Machines\test-winxppro-sp2` if you have followed along with the example. Once you have that selected, it will be added to your inventory and displayed on the main pane.

Once the machine has been added to your inventory, double-click it to make its tab appear in the main pane of the VMware Server Console. Select Edit virtual machine settings from that tab and choose the Options tab up top in the Virtual Machines Settings window. This is shown in Figure 6-2.

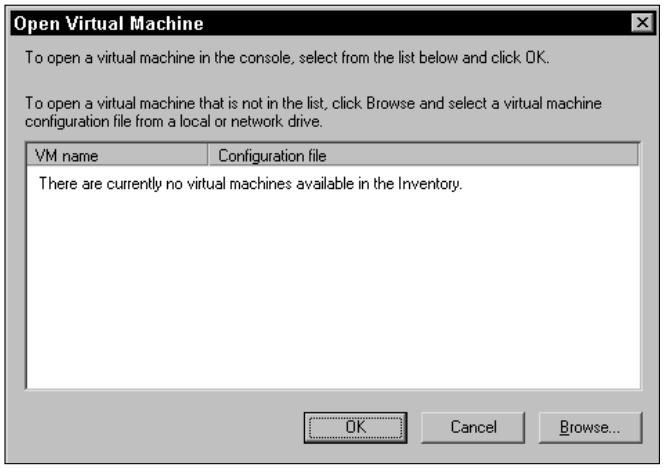
Open Virtual Machine
To open a virtual machine in the console, select from the list below and click OK.
To open a virtual machine that is not in the list, click Browse and select a virtual machine configuration file from a local or network drive.
VM name
Configuration file
There are currently no virtual machines available in the Inventory.
OK
Cancel
Browse...

Figure 6-1

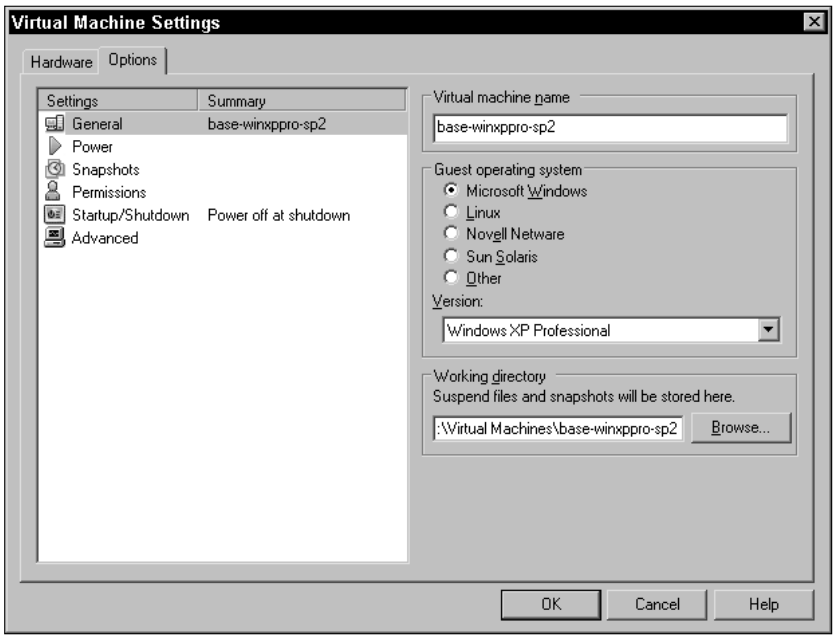

Figure 6-2

It's pretty self-explanatory from here—change the display name shown on the Options tab to reflect the new virtual machine's name, in this example `test-winxppro-sp2`. Select OK and you're done. The base image has been copied over, added to the inventory, and adjusted to fit your naming scheme and enter your production environment.

> *Remember that if you did not use Sysprep during the creation of your base images, you will need to generate a new SID for the newly created image. This is a requirement that must be met for your Windows-based images. See Chapter 4 for more information on Sysprep and an alternative method of generating new SIDs.*

Recycling Your Testing Images

There will be some images that you use over and over again. The most common ones will be your testing images. Because software testing normally involves installing an application, using it, and then either uninstalling or slicking the testing machine in extreme cases, time becomes a serious factor. This common flow plays right into two of VMware Server's strengths, nonpersistent disks and snapshots.

Both nonpersistent disks and snapshots were defined and discussed in Chapter 5.

Independent disks, as was covered in Chapter 5, come in two flavors, persistent and nonpersistent. Persistent disks write their changes to disk immediately, whereas the nonpersistent disks write to disk only temporarily and discard the information once the virtual machine is powered off. This behavior makes the nonpersistent disk a very valuable tool for testing machines.

One of the problems with a testing machine is that once you install your application to test, there's always a chance that something will get left behind. Installers especially are not perfect if that is what you happen to be testing at that moment. Any stragglers from the previous install could seriously affect the outcome of all subsequent test runs. With the use of a nonpersistent disk the worries are gone. Simply start up the testing image and test your application. Want a fresh testing machine? Just power down the virtual machine, and it immediately reverts to its state prior to the first test. Quick and simple.

To set up your virtual machine to use nonpersistent disks, follow these steps:

1. Open the machine in the VMware Console and select the virtual machine settings.
2. Choose the hard disk from the left-hand pane and click the Advanced button located in the settings pane to the right. At this point you should see a screen like Figure 6-3.

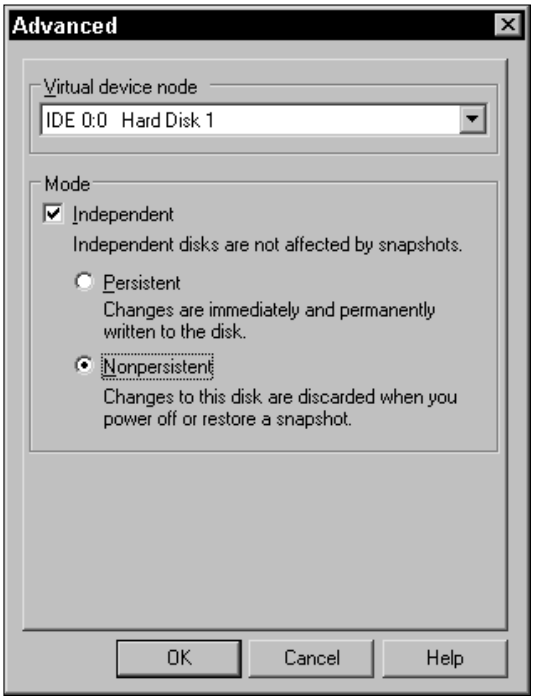

Figure 6-3

3. Select Independent disk and choose Nonpersistent. Click OK until you get back to the main console screen and you are set.

Go ahead and test it out. Start up that virtual machine, make a change, and then reboot. The change should be gone when you fire it back up. You can hardly beat it.

> *If you are using a base image that you have not previously run, you will need to start up that virtual machine and make it through the Sysprep setup before switching the disk to nonpersistent.*

One downfall of the nonpersistent disk and a situation in which you would be unable to use one is where either your install or a prerequisite install requires a reboot. Obviously this test situation will utterly fail if you are using a nonpersistent disk because as soon as the machine cycles, the nonpersistent disk loses the information. You would never make it past the first stage of the installer. There is, however, a nice alternative and it is the snapshot. If you remember, the snapshot allows you to take a picture, if you will, of the current state of the machine. You cannot take snapshots of independent disks; however, as in the case of the preceding scenario where you have a multistage install to test, an independent disk is not the right choice. To address the preceding example, you would want your disks to not be independent and then you would take a snapshot of the virtual machine state prior to the test. This would still allow you to revert back to a state prior to the test just as the independent disk would, but you would gain the flexibility of maintaining state across a reboot.

Creating an ISO Image Library

I talked briefly about ISO images in Chapter 4 and how you can increase performance during your operating system installs by using an ISO image as opposed to the actual physical CD or DVD disk. So, why would you need a library of them? Well, you don't absolutely need a library of them; it is entirely up to you. However, convenience and performance are your driving factors here, and there are some definite advantages to putting a little time into the ISOs:

- **Convenience** — As a brief recap, an ISO image is an uncompressed file that contains the data and structure of a CD-ROM or DVD-ROM disk. This ISO is a single file that you can mount in VMware to simulate the actual CD or DVD disk being in a physical drive. The benefit here is that instead of swapping physical disks out of that overstuffed binder on your shelf, you simply have to point to a different file on your host server. Each file in and of itself is the disk; there's no difference as far as VMware Server is concerned. If you remember from Chapter 4, selecting an ISO file for your virtual CD-ROM drive to use is simple. Open your virtual machine and from the main tab select the virtual machine settings. On the left select the CD-ROM drive, and you'll be presented with the settings shown in Figure 6-4.

  Just select the button beside Use ISO image and click the Browse button to locate the image file you are interested in. VMware Server will emulate the CD-ROM drive to your virtual machine. Your guest machine will never know the difference. I'll give you a better example of this setting when I talk about creating a new ISO image in the next section.

- **Performance** — ISO images provide a significant performance boost to media access via the virtual CD-ROM drive in your virtual machines. By accessing the data from an ISO rather than the physical disk, you get the benefit of the speed. Hard disk access is by far faster than a spinning CD in a drive. It's a performance boost that you will notice without a doubt.

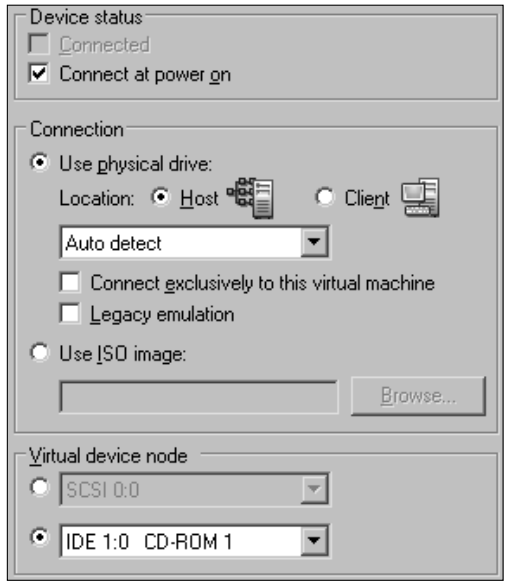

Figure 6-4

An ISO image is based upon a file standard. That means if you are creating your ISO images on Windows-based systems, you should be able to utilize that image on a Linux-based system as well.

Creating an ISO Image

A variety of software programs are available to help you make ISO images; however, I have a personal favorite, WinImage. WinImage can be obtained from `www.winimage.com`. For the purpose of this example, the shareware version is just fine. Go ahead and download and install WinImage. I want to walk you through the creation of an ISO image from a standard Windows XP Professional installation disk, though this will in no way be a walkthrough of all the options available within the software. I leave you to explore those on your own.

1. Start up WinImage via the Start menu or icon on your desktop.
2. Insert the Windows XP Professional installation disk into your CD-ROM drive and select Create CD-ROM ISO image from the WinImage Disk menu.
3. Next you will be asked to select the CD-ROM or DVD-ROM drive the disk is in, as well as provide a name for the new image.

   Just select the drive that contains the disk you want to image. In the name blank I use the same format technique as I demonstrated with image files. The `<type>-<operating system>-<patch level>` naming scheme works here as well. Consistency is the key to organization. I tend to create a directory just for ISO images. I tend to place new images into an ISOs directory on the local disk. There is no rule that says you have to keep them separate from your virtual machines, but I recommend you make a nice tidy little spot away from your images, archives, and running virtual machines to keep these files. One caveat to mention, however, is that you will want to store the ISO images on the host server itself. There is little benefit in using an ISO image if you are dragging it across a slow network link.

You will need to make sure the directory in which you place your ISO image already exists prior to executing the command. Failure to do this will result in an error from WinImage. If this happens simply create the directory and repeat the preceding steps.

4. Once you select your drive and take care of the name, click OK. You will receive a nice progress bar for the operation.
5. Once the operation is complete, WinImage displays the contents of your ISO image.

WinImage is not only a great tool to create your ISO image files with but also to edit and view the contents of an ISO image file. If you go ahead and close WinImage for now and browse to the location in which you saved the ISO image file, you will find that WinImage has become the default handler for the `.iso` file extension. Just double-click any ISO file and WinImage will display the image and allow you to edit and manipulate the contents.

Now, take a quick look at what WinImage provided. If you happened to use the same directory structure I did, open My Computer and browse to the `C:\ISOs` directory, as shown in Figure 6-5. If you used a different directory, browse to it instead.

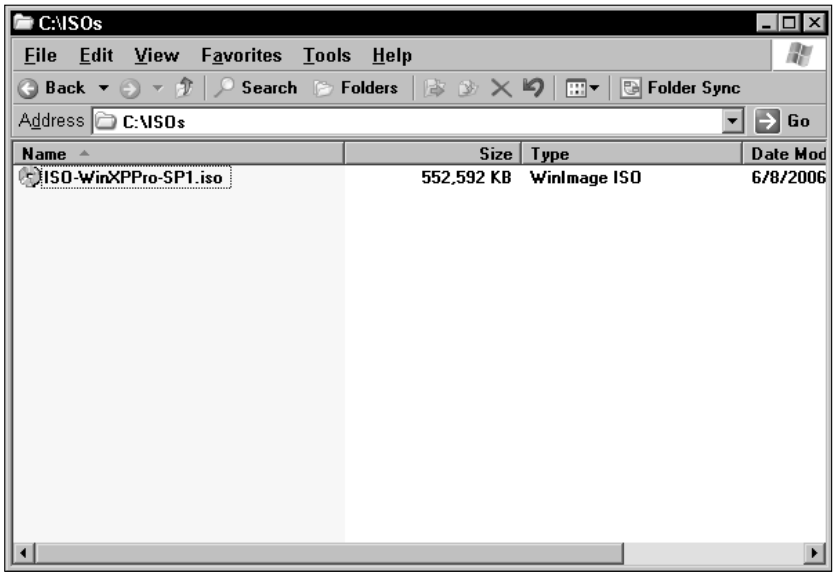

Figure 6-5

Using an ISO Image

Okay, so now that you have your first ISO image, how do you use it? Just for the sake of example, create yourself a new Windows XP Professional virtual machine. Once you finish the wizard, open the virtual machine settings from the main tab in the console and select the CD-ROM drive, as in Figure 6-6.

Select the Use ISO image option under the Connection group and browse to your newly created ISO image file. You will notice in the preceding figure I chose the Windows XP Professional SP1 ISO I just created. Click OK and start up your virtual machine. The ISO image file is attached to your virtual CD-ROM drive and, if no operating system is present, the virtual machine will boot from the virtual drive. The virtual machine can see no difference between an ISO image and a physical disk.

For example, imagine your host server is sitting in the server room, and you are configuring new images from the VMware Server Console on your office desktop — a very likely setup. As you are installing software on, say, a new development image, you need the disk for Visual Studio 2005. You could, through the power of VMware Server, connect the virtual machine to your local DVD drive on your office workstation, but this is a performance hit because the DVD data must traverse the network. You could also connect the virtual machine to the DVD drive located on the host server itself but this requires you to get the disk and walk to the server room and put it in the drive. It solves the performance hit, at least the network part of it, but you still have to do some legwork. What if you had taken the time to just create an ISO image of the DVD in the beginning? No disk is to be found, nor is any legwork involved. Simply open the virtual machine settings and point it to the ISO image needed and you're off.

Creating ISO Files in Linux

Creating ISO files in Linux is very simple. There are, of course, several different ways to accomplish this with various tools and packages available on Linux. The following method uses two commands that are generally on all Linux distributions, `dd` and `cksum`.

First you would obviously insert the CD-ROM into the drive that you want to make an image of. Most modern Linux distributions mount this disk for you. Where the disk is mounted to depends on your distribution. In the case of Red Hat the CD-ROM is mounted into the `/mnt/cdrom` directory.

After you insert the disk, make sure you are in the `/mnt/cdrom` directory (or change into it if you are not) and verify you can see the contents of the disk. If not, issue the following command as root:

```
mount /mnt/cdrom
```

This will mount the CD-ROM into the `/mnt/cdrom` directory.

Next, you simply need to copy the contents of the CD-ROM into a file ending in .ISO. You can do this with the following command:

```
dd if=/mnt/cdrom of=/image.iso
```

The parameters are simple. `if` refers to the input file, in this case the CD-ROM itself, and `of` refers to the output file that you save into `/image.iso`.

Once you execute this command, a new ISO is created in the root of the drive named `image.iso`. To ensure the image is an accurate ISO of the CD-ROM, you can run a manual checksum on each with the following:

```
cksum /mnt/cdrom

cksum /image.iso
```

Compare the checksum values returned and ensure they match. If so, you have a valid ISO file ready for use in VMware Server.

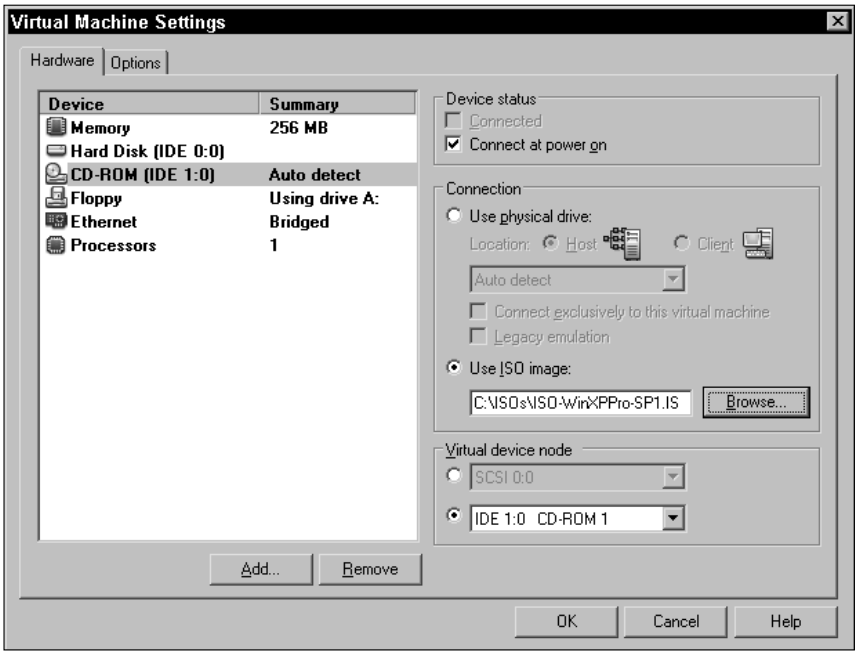

Figure 6-6

What if the virtual machine is already running, and you need to perform a disk change? Simple, if you remember from Chapter 4 I talked about the little hardware status icons in the bottom-right corner of the console. Figure 6-7 shows these icons as a reminder.

Figure 6-7

If you right-click or double-click the CD-ROM icon, you can change the ISO image file it points to on the fly. A quick and easy disk change that doesn't require you to grab that overstuffed binder off the shelf.

Summary

In this chapter I wanted to take a few moments to follow up on the previous two chapters and provide you with some direction to better organize and maintain the image libraries you create. This chapter covered the following:

- Organizing and maintaining your image library
- Preparing images from your base library for use
- Recycling your testing images to avoid unnecessary shuffling of new virtual machines
- Creating an ISO image library of commonly used CD-ROM and DVD-ROM disks

Having an organization plan for your virtual machine image library is paramount. The purpose here is to adopt virtualization on a grand level and with that purpose come lots of things to manage. A set of best practices saves you in the long run.

Preparing your base images for production use is not a difficult step but nonetheless a vital one. The important thing to remember is to always stick to your naming scheme. Keeping your machines unique and organized can save you. Remember to copy your base images, instead of moving them, and to rename both their directory and display name within the VMware Server Console.

By recycling your testing images through the use of nonpersistent independent disks you can decrease the number of testing images in use and increase your productivity time by foregoing the task of creating or moving around your images to support various testing scenarios.

Finally, creating the ISO image library can be both time saving and performance boosting. By extracting your more commonly used CD-ROM and DVD-ROM disks to ISO image files you make the creation and everyday use of your virtual machines easier. No more passing around the MSDN binder to your team. Simply image the disks and away they go.

From this point forward you are going to see a dramatic change in the chapter content. I'll be moving from the general operation of VMware Server into a more distinct path of process integration. The next few chapters introduce the VMware APIs, or programming interfaces, that you will use toward the end of this book to integrate VMware Server into your development process in a more automated fashion.

7

Introduction to the Scripting APIs

In today's digital world the ability to script common repetitive tasks is one of the keys to increasing productivity and integrating your workflow process.

A perfect example of this from a development standpoint is a software package that checks out your code from source control, builds it, and then runs a series of unit tests on it. The software package in this example does not contain all the functionality to perform those tasks itself. It accomplishes it through a series of scripts that interface with other services, like source code control and your unit testing framework. Without the ability to script these actions your software would remain disjoined and require some level of manual intervention in order to accomplish the goal. With the ability to script the actions, however, it becomes a well-oiled machine that can easily jump the boundaries between separate applications to perform a common set of tasks.

> *As you will learn in later chapters, the software I just described exists in Visual Studio Team System and CruiseControl.NET. Because they offer you the ability to automate actions such as nightly builds and unit testing, they should also be able to interface with your virtual machines. Well, they can, and you will do this very thing in the coming chapters.*

For you to integrate VMware Server into your entire development process you need to understand what methods VMware has provided for you to automate actions such as powering on and off your virtual machines and many others. This automation can be achieved through the scripting application program interface (API) that VMware has so graciously provided.

What Can You Automate?

In life there is always a limit to what you can do, and that holds true for the VMware scripting API. However, you will find the majority of what you need to accomplish for automation or software integration can be done. The following are just some of the common tasks you can accomplish with VMware server via your scripts:

- Powering on, off, rebooting, and suspending your virtual machines
- Listing the virtual machines available and in inventory on the VMware Server
- Viewing the current state of a virtual machine
- Sending and receiving information from the guest machine
- Viewing any errors associated with your virtual machine
- Answering questions and clearing errors on your host machine
- Connecting and disconnecting virtual hardware devices for specific virtual machines
- Modifying certain configuration variables for each virtual machine

As you can see from the preceding list, the API is a very powerful tool in your arsenal. You will be literally amazed at the functionality you can obtain through a simple script and this interface.

As an example, say you have a testing machine named `test-winxppro-sp2` in inventory on your host server. By way of the scripting API, you could write a script that connects to the host server, searches for the virtual machine named `test-winxppro-sp2`, queries the status of that machine and if not active powers on the machine, retrieves the IP address of that machine once powered on, and copies a file via the network to its desktop. Pretty slick isn't it?

You can take this a step further. Say, for instance, you have a team of Quality Assurance (QA) inspectors that verify and test your nightly builds. Utilizing Visual Studio Team System you can automate the nightly build and run unit tests, but QA wants to see it on an actual system to test the installer and usability aspects of the application. Now, you could simply drop the nightly build payload into a directory that QA has access to; then, once they arrive in the morning, they can take that build, fire up the virtual machine, and test away. But can you make this better? Yes, through the scripting API with VMware Server you could fix it so that after the nightly build finishes a script runs that starts up a series of virtual machines for QA to test on, copies the installer up to the machines, and then e-mails each member of QA with a specific machine assignment ready to go for them. When QA arrives you have already prepared and configured their testing environment for the day. All they have to do is launch the VMware Server Console and test away.

What you can accomplish with this is pretty much bound only by your imagination. The ability to power cycle and control your virtual machine's configuration is made rather simple through the API. Other, more advanced topics, like passing data between the virtual machine and host server, are a little more complicated but only because it takes a deeper understanding of the scripting API and not that it cannot be accomplished. In the coming chapters I go over the three types of scripting available to you and provide some sample code that you can easily use to witness the power you can harness.

In Chapter 11, I will be doing something very similar to the example I just explained.

Supported Platforms

The scripting APIs from VMware come in three distinct but functionally equivalent varieties: Perl, COM, and the Programming API (which utilizes C). Each of the three APIs provide you with a similar set of functionality and exist separately only to give you the ability to utilize the scripting language you are most comfortable with.

> *The Programming API offers an expanded set of functionality over Perl and COM. This API is a new offering from VMware so its feature set is still in flux. I cover its expanded feature set in Chapter 10.*

Now, before I get into the specifics of what API is available to what operating systems, it should be noted that the API in use does not have to reside on the host server. You can download just the APIs for both the Windows and Linux platforms independent of the server package. Most likely you will have the API installed on your local workstation during the initial phase of implementation, but in the case of integrating other software packages to work with VMware Server, you will need to take that server's operating system into account. In other words, if you are integrating Visual Studio Team System with VMware Server, you will want to utilize an API that is available for the Windows platform. I explain your options in the next few sections.

VmPerl

The VmPerl API can be installed and used on both Windows- and Linux-based machines. VMware packages its own Perl interpreter, which allows it to be independent of the version of Perl already installed on the local host server.

> *The scripts written in this book include a declaration for Perl on Linux as well as on Windows so the scripts can be portable between host operating systems.*

VmCOM

The VmCOM API is a Windows-only technology and therefore is available only on Windows-based host servers or workstations. COM, which stands for Component Object Model, is a Microsoft technology that facilitates communication between software applications. Though COM has begun to disappear from mainstream Windows-based development as of late, the VmCOM API does a good job of separating the implementation from its functionality. Even though the desire to develop in COM has lessened, you should not shy away from leveraging the COM API for that reason. COM will continue to be supported under Windows for many years to come.

The Perl Interpreter

Perl is an interpreted language. What this means is that the scripts you write in Perl remain in their text form and are parsed and interpreted at runtime by Perl itself. This is in comparison to a language like C where the source files are compiled into a binary format prior to use.

Every script you write in Perl must reference the targeted interpreter at the very beginning of the file. The slang term for this declaration is the *shebang*; however, it is simply `#!` followed by the path to the interpreter. An example of this for the Linux platform would be `#!/usr/bin/perl`.

The beauty of VmCOM is that it allows you to write scripts that interact with VMware Server in a much larger variety of languages. VmCOM can be leveraged with VBScript or JavaScript facilitated via the Windows Scripting Host as well as some more robust languages like C# and VB.NET. Simply put, VmCOM opens a lot of doors for you in terms of integration, but its major drawback is that it is available only to Windows-based servers and workstations. The functionality between VmCOM remains the same in terms of interaction with VMware Server.

Programming API

The Programming API, also known as Vix, is the newest in VMware's API offerings and is currently available only for VMware Server. It offers an expanded set of functions as opposed to the previous two offerings but adds a layer of complexity for the average or even above average user.

> *VMware plans on expanding the Programming API to include support for other products in its virtualization line in the near future. Refer to the VMware web site (*`www.vmware.com`*) for up-to-date information regarding a specific product's compatibility with the Programming API.*

If you are familiar with C programming and you are attempting some rather adventurous projects, this is the API for you. It is thick and robust, but it is new and growing as well. One disadvantage at the moment is that it only supports use in other applications written in C.

If this hasn't scared you off there are two header files you need to include in your C application as well as a few DLLs you need to link when you compile. I provide some brief examples when I cover the Programming API in Chapter 10; however, I don't spend a great deal of time on it because I believe most will find all the functionality they need in the other two APIs. Vix is really designed for development-level integration and advanced projects and, therefore, is beyond the scope of this book for anything other than an introduction.

VmPerl vs. VmCOM vs. Vix

Who doesn't like Perl? I love it and tend to gravitate toward it on most occasions; however, although these APIs have similar sets of functionality, there are some advantages and disadvantages to each:

- **VmPerl** — VMware ships all versions of its VmPerl API with a copy of a Perl interpreter included. In the grand scheme of things that makes it a little easier to use because there are no external requirements for you to start cranking out scripts. Perl is easy to learn and a rather robust language in and of itself. You won't find yourself spending weeks or even months trying to learn the language. The beauty of Perl is that you only need to learn the basics and you are off and running. A deep understanding is not required.

  One key feature that makes it a superior choice over VmCOM is portability. You can write scripts utilizing the VmPerl API that are completely portable between Windows- or Linux-based VMware Servers. This requires a few extra lines of code per script but it is well worth it. I talk more about that when I start showing you some sample scripts in later chapters.

  There are, however, a couple of disadvantages to the VmPerl API in my opinion:

  - First is that Perl is unfortunately not a common skill in the Windows administration world. This is by no means a shot at Windows administrators; it's just an acknowledgment that Perl is not something most end up dealing with.

- Second, with the VmPerl API you write your scripts in Perl; that's the only choice. This would normally not be considered a disadvantage but lined up next to the VmCOM API, where language choice is a key benefit, it is.

- **VmCOM** — VmCOM is based upon the COM method of application integration. COM is a programming interface and not a language in itself. An interface is simply a method in which two disjoined pieces, in this case VMware Server and the script, can communicate. By utilizing this interface you can harness the COM objects through scripting languages like VBScript and JavaScript as well as the full blown, latest and greatest language, like any of the .NET languages. As the examples start to flow in upcoming chapters, you will notice a striking similarity between the usage of both the VmPerl and VmCOM APIs.

  There are two key advantages to leveraging the VmCOM API over VmPerl:

  - The first, as I alluded to previously, is the number of scripting languages at your disposal. With VmCOM you are not stuck with a single language like Perl; you have multiple choices at your disposal. Not familiar with VBScript, but you know C#? No problem — just develop your scripts as a console application in C#. I could continue but I think you get the point — you have a choice.

  - The second advantage, and probably not so much a deciding factor with the level of scripting you are doing here, is the ability for your scripts to have a graphical interface. There is absolutely nothing that keeps you from referencing the VmCOM API in a regular GUI-based program. Though this is a bit of overkill for most of your purposes here, it is an important distinction to point out because it does not exist as an option with Perl.

  Of course there are a couple of disadvantages to the VmCOM API:

  - First is platform portability. VmCOM, or any COM component for that matter, is strictly Windows-based. You cannot develop scripts against VmCOM and expect them to port over to your VMware Server install sitting on a Linux host. If you choose VmCOM you by default choose Windows.

  - The second is the Component Object Model itself. COM is fairly old and, therefore, in favor of newer .NET technology present in Windows-based development, could see drastic changes in support for the future. Though there are no plans by Microsoft that I'm aware of to give COM the axe in terms of platform support, it is possible that your options for leveraging COM in the future could become limited.

- **Programming API (Vix)** — The final option, the Programming API, is really a little new for me to have formed an opinion on. VMware has taken great steps in providing you with a richer, more complete package with it; however, it currently lacks the flexibility present in the other packages. I outline the specific functions, properties, handles, and error codes in Chapter 10 as well as provide some basic example code to leverage it, but because the API only supports scripts written in C at this time, I will not be providing any code during the integration scenarios in Chapter 11 and 12 against the Programming API. The Programming API is available for both the Windows and Linux platforms and, therefore, could gain a distinct advantage over VmPerl in the near future.

I've laid out the basic advantages and disadvantages as I see them between the three APIs. The choice really depends on your comfort level as well as your platform requirements. VmPerl and VmCOM are extremely flexible and get the job done in the exact same way. The VmPerl API comes with everything you need, is platform independent, and simple to use; however, if you have never used Perl, it is a bit of a

hurdle. The VmCOM API offers several options for scripting and development languages, provides the same functionality as its Perl cousin, and adds the ability to leverage a graphical interface if you so choose. It is, however, strictly a Windows-based option. Lastly, the Programming API is really an infant as far as API choices for VMware Server and as such might not be the best choice at this time. The Programming API will definitely attract some serious developers and system integrators in the months to come, but for streamlining your development process it is just not the best tool for the job. At least not yet.

> **I feel as though I should stress again that just because your server is Linux-based doesn't mean you must use the VmPerl API. Remember your scripts do not have to reside on the host server itself. They can be run from any system that has access to the server. If your host server is Linux but your process integration server, like Visual Studio Team System, resides on a Windows server you can use VmCOM if the scripts are housed on the Windows machine. The choices are abundant.**

The vmware-cmd Utility

The `vmware-cmd` utility allows you to execute various command-line–based actions against your virtual machines. Through the use of this command-line tool you can achieve almost all of the same functionality that you can by scripting against one of the APIs. The downfall is that using `vmware-cmd` requires you to perform actions in single steps with little or no error control.

An example would be if you attempt to use the `getstate()` function with `vmware-cmd`. A command such as the one that uses `vmware-cmd` on a Windows host to get the current status of a virtual machine named `dev-winxppro-sp2` can be seen in the following example:

```
vmware-cmd C:\virtualmachines\dev-winxppro-sp2\dev-winxppro-sp2.vmx getstate
```

> *If the path to your virtual machine configuration file is in a directory that contains spaces, be sure to remember to enclose that portion of the preceding command in double quotes.*

If this machine is not currently in inventory, this command will throw an error stating that the virtual machine in question was not found or not registered and `vmware-cmd` will exit.

The same command can be executed with error handling in a Perl script. By using the VmPerl API and a Perl script you can catch the error thrown if the virtual machine is not registered and rectify the situation, all within the same script. I provide such a script later in Chapter 9 that demonstrates just that.

The `vmware-cmd` utility is located in the `/usr/bin` directory on Linux servers and the `C:\Program Files\VMware\VMware VmPerl Scripting API` directory on a Windows-based server. The interesting part is if you take a closer look at the `vmware-cmd` utility on Windows you will notice it is simply a batch file that calls a pre-existing Perl script named `vmware-cmd`. Although you could script the `vmware-cmd` utility to perform a series of commands against your server, it is really designed for quick one- or two-step functions. I recommend you leave the heavy lifting to one of the APIs.

The vmrun Utility

With the introduction of the Programming API, or Vix, VMware has added another utility called `vmrun`. The `vmrun` utility is very similar to the `vmware-cmd` utility described in the preceding section; however, its functionality is severely limited. With `vmrun` you can list the virtual machines currently registered with the local server and power cycle them, and that is pretty much all you can do.

As the Programming API matures, the `vmrun` utility will most likely gain some of the functionality already present in the `vmware-cmd` utility. At the moment, however, it is good for little more than a quick power on or off of an already registered virtual machine.

The VMware SDK

"Why can't I use the new VMware Software Development Kit (SDK)?" you ask. For those of you who have surfed around the VMware site and had the SDK catch your eye, you may very well ask this question. The answer is simply that maybe someday you can, but for now, you can't. And here's why: The VMware SDK is essentially an extension of VMware's virtual infrastructure product, VMware Infrastructure. VMware Infrastructure allows for the management of multiple VMware Servers over your entire infrastructure. This is accomplished through a central web management site very similar to the web management interface you installed with VMware Server, only much more capable. The VMware SDK is an extension of this in that it provides an interface in the form of a web service. By leveraging the power of web services, VMware has opened the door to deep application integration with its server products.

VMware Infrastructure is not free like VMware Server, nor, as of the time of this writing, does it support VMware Server. If centralized management is a desire of yours, I encourage you to keep an eye out for VMware Server support under VMware Infrastructure. Until then the SDK is of little use to you as far as VMware Server is concerned. In terms of automating your development and testing environment, the APIs are more than adequate anyway.

Summary

This chapter introduced you to the options available as you start to plan and implement the integration of virtual machines into your development process:

- You got a brief introduction into just what the APIs can do for you. From registering new machines and power cycling to retrieving information from the guest machine, the overview provided should at the very least give you a taste at just what you can accomplish.
- Supported platforms are important when it comes down to what API you choose to develop against because where the script ultimately resides will be based upon your environment. An understanding of your options available per platform will aid in this decision. You don't want to write your scripts in COM just to decide later you wish they could live on your Linux server.
- The VmPerl, VmCOM, and Programming API sections provided you with a brief introduction into the technology that led to a comparison of the strengths and weaknesses of each, with an eye toward helping you choose what might be the best path for you to follow in system integration.

- The `vmware-cmd` and `vmrun` utilities are provided by VMware to assist you with quick, one- or two-command operations against your virtual machines. These two utilities are in no way able to handle some of the more advanced operations you will be performing in the pages to come but offer the ability to make some quick command-line changes. They are especially useful in a Linux environment where command-line management takes on a bigger role over its place in the Windows world.

This chapter has given you just enough of a glimpse into the APIs available so that you can focus on the specifics of how to implement them in the chapters to come. Over the course of the next three chapters I provide you with a programmer's reference guide of sorts that you can easily refer to as you start building your integration scripts.

8

Using VmCOM

Now it's time to get your hands dirty. Over the next three chapters I walk you through the various API options available to you as well as provide some basic sample code. Though you will most likely focus on one particular API in the end, I encourage you to go through each of the chapters in order to fully grasp the options available. Who knows, you may decide the VmCOM API is easier to leverage than VmPerl even though you have a special place in your heart for Perl.

VmCOM is the first of three APIs I am going to cover. The VmCOM API is a C Library, written by VMware, that is based upon the COM, or Component Object Model, interface. Simply put, VMware designed a library to communicate with its server products and you can interface with it via COM. Now we don't care how the library works, we are merely interested in what it exposes for our use. If you are not familiar with object-oriented programming, or OOP, then I should tell you that this chapter is not a thorough primer for OOP, though I do take a few moments in the next section to give a quick and dirty primer to the terminology utilized throughout the next three chapters.

> *There are several good references out there to help you understand the basics about objects, methods, enumerations, and so on, so I won't cover them in detail here. This knowledge is not needed, however, in order to utilize the APIs.*

A Quick OOP Primer

First there was an object. An *object* in basic terms is nothing more than just a representation of a physical object, your server for instance, in code. An object is the primary means by which you will define and manipulate your servers and virtual machines. You create an object, perform various actions against it, you destroy the object. That's pretty much the workflow involved.

The OOP rule of thumb is that an object has state; if it does not then it is not, or should not be an object. *State* refers to a characteristic of an object. A car is an object because it has color, velocity,

and so on. That a car has a color of blue and that it has a velocity of 0 are both examples of the state of the car object. I get more into how this relates in the following paragraphs.

So, I mentioned actions. In order to actually perform actions against your server you have to use something called a method. A *method* is an action performed against its parent object. Using the car object mentioned previously, say you want to start it. Methods can be used by referencing the parent object, followed by a period, and ending with the method name. Assuming the method to start your car object is called `Start()`, an example of this would be `Car.Start()` where `Start()` is the action, or method, performed against `Car`.

How about the state of an object, its size or color? This is defined through what is called a property. A *property* defines the state of an object. If you extend the preceding example, you can say a car has various states associated with it. `Running`, `Stopped`, and `Color` are all states of your car. To set the color of the car to blue, you would say `Car.Color = "Blue"`.

Now, in the preceding example you set the property `Color` to `Blue`. This is referred to as a *set*. You *set* the property `Color` to the color `Blue`. The opposite of this is called a *get*. To *get* a property you query it, `Car.Color`, and it returns the color `Blue`. The actual method in which you perform the two is different but the basic concept is there. This will become clearer once you start to see some example code.

Finally, there's the enumeration. An *enumeration* is a list of values commonly associated with a list of strings, or names. An example of this might be the different colors available for your car. The enumeration `Colors` defines a list of available colors `{Blue, Red, Green, Yellow}`. The `Color` property listed previously can be made to only accept colors from the `Colors` enumeration. Setting a property to only accept values from a specific set, like an enumeration, is something that the API developer does, not you. Your only concern is to understand what enumerations are and apply them where necessary.

As I mentioned before this is not meant to be a lesson in OOP but simply a quick overview of the types of items you will find within the API. This information in combination with the reference and example code should put you on your way to writing your first script for VMware Server.

Windows Scripting Host

Because this chapter is talking about the VmCOM API, it is, of course, limited to the Windows platform. The examples in this chapter are in VBScript utilizing the Windows Scripting Host, or WSH. Each example consists of two parts: the actual script file, which will end in `.vbs` for VBScript, and the associated Windows Script File that ends with `.wsf`. The `.wsf` and `.vbs` script files must reside within the same directory.

You can execute the scripts using the WSH with the following syntax:

```
cscript //nologo <name of script>.wsf
```

The contents of both the `.wsf` file and the script file are included with each example.

> *The command-line switch of* `//nologo` *is optional; however, its use tells the Windows Scripting Host to not output its version information at the beginning of every script. This allows for a cleaner command line while executing your scripts.*

Objects

The VmCOM API contains five major objects that represent various tangible pieces of your VMware Server environment. The following is a list of the objects exposed in the VmCOM API:

- `VmConnectParams`
- `VmServerCtl`
- `VmCtl`
- `VmQuestion`
- `VmCollection`

Each of the preceding objects has a specific purpose and a set of methods and properties associated with it. Over the next few sections I explore each and provide a list of the methods and properties along with their usage.

> *The following object reference draws on and adapts information from the official VMware Scripting Guide. I believe the following content will serve you well as a quick reference because my focus here in this book (as opposed to that in the documentation) is directly against VMware Server.*

VmConnectParams

The `VmConnectParams` object is sort of an access card object to the `Connect()` methods you will see for `VmServerCtl` and `VmCtl`. This object exposes four properties that allow you to set and get a host name, port, user name, and password. Its sole purpose is to store these values so you can use them to open a connection to your VMware Server or virtual machine through the use of other objects. This object is listed first in the reference because it is most likely the first object you will create in each and every script you write.

> *Remember how your console connection was encrypted and showed that little lock icon in the bottom-right corner? If your VMware Server is configured to encrypt communications between itself and its consoles, the information passed by this object to the server is also encrypted.*

The following table lists and describes the properties of this object.

Name	Type	Access	Description
`Hostname`	string	Get/Set	Gets or sets the name of server you wish to connect to. If the host name is not given, or left undefined, the server is assumed to be on the local host.
`Port`	integer	Get/Set	Gets or sets the TCP port number to use when connecting to the server. If left undefined this value will default to 0 (zero), which in turn is read as the default port tcp/902. If you use a nonstandard port number enter it here. A negative number will default to the default port tcp/902.

Table continued on following page

Name	Type	Access	Description
`Username`	string	Get/Set	Gets the user name that is authenticated or sets the user name you wish to authenticate to the server with.
`Password`	string	Get/Set	Gets or sets the password for the user defined in the `Username` property.

Because this object in and of itself does nothing more than provide an "access card" to the server or virtual machine, there is little use in an example as of yet. There are plenty of examples on how to use this object in the sections to come.

VmServerCtl

This object is at the center of everything you do. More often than not, unless you know the specific name of the virtual machine you wish to manipulate, you will be creating a new `VmServerCtl` object.

This object exposes one property and three methods. Their functions are outlined in the tables that follow.

> *Starting with this object you may notice the API documentation lists more properties or methods available than I list here. Most often this is because the API can interface with ESX Server, another VMware product, and it has methods and properties specific only to that version. I will document only the API functions that are relevant to the version of VMware you are using, in this case VMware Server.*

Properties

Name	Type	Access	Description
`RegisteredVmNames`	VmCollection	Get	Gets an object of type VmCollection containing strings specifying the configuration file names of the virtual machines currently registered on the server you are currently connected to.

Methods

Name	Parameters	Description
`Connect()`	`VmConnectParams`	This method connects the parent object to the VMware Server specified in the `VmConnectParams` object. Refer to the `VmConnectParams` object documentation for more information. The total number of connected objects cannot exceed 62 or an exception will be thrown.

Name	Parameters	Description
`RegisterVm()`	`vmName` as string	This method registers a new virtual machine with the server you are currently connected to. `vmName` is a string that refers to the configuration file name of the virtual machine you wish to add to inventory.
`UnregisterVm()`	`vmName` as string	This method unregisters a virtual machine currently in inventory on the server you are connected to. `vmName` is a string that refers to the configuration file name of the virtual machine you wish to remove from inventory. This does not delete the virtual machine files.

Example

Now for an example — in this script I show you how to set up your `VmConnectParams` object, connect to a virtual machine using the `VmServerCtl` object, and register a known virtual machine.

1. First off you need to create your WSH file, the one that ends in `.wsf`. This file tells the WSH which COM library to load and what script file to parse and execute. Open Notepad, or your preferred text editor, and enter the example code that follows.

This example assumes that you have a working VMware Server installation and at least one virtual machine configured but not currently in inventory.

example1.wsf

```
<job id="Example1">
  <reference object="VmCOM.VmCtl" />
  <script language="VBScript" src="example1.vbs" />
</job>
```

Save this file as `example1.wsf` and place it in a location where you will be keeping your scripts for this example.

2. Next you need to create the actual script itself. Once again open your favorite text editor and enter the following code. This script assumes a virtual machine exists on your server with a configuration file named `dev-winxppro-sp2.vmx` and is located within the `C:\VirtualMachines\dev-winxppro-sp2` directory. This script also assumes that the virtual machine listed is not already registered in inventory.

   In a real production script you would most likely not hard code such things into your script, but for the purpose of this example it works just fine.

example1.vbs

```
' First we create the VmConnectParams object to hold our authentication data
Set connect_params = CreateObject("VmCOM.VmConnectParams")

' Setup the connect_params to use the specific server and user we choose
```

(continued)

example1.vbs *(continued)*

```
' Remember, if the port is not specified then it uses the default of 902
connect_params.Hostname = "vmserver-01"
connect_params.Username = "administrator"
connect_params.Password = "P@ssword!"

' Here we create a new VmServerCtl object to represent our server
Set server = CreateObject("VmCOM.VmServerCtl")

' Call the Connect method and pass our authentication info set above
server.Connect connect_params

' Run the RegisterVm method to add our vm to inventory
server.RegisterVm "C:\VirtualMachines\dev-winxppro-sp2\dev-winxppro-sp2.vmx"
```

3. Save this file as `example1.vbs` and place it in the same location as the `example1.wsf` file you created previously.

So, you can now kick it off and see what happens. You start the script by opening a command prompt and typing the following at the command line within the directory you saved the scripts to:

```
cscript //nologo example1.wsf
```

If all goes well the script will exit without warning. To verify the script did in fact register the virtual machine into inventory you will need to open a VMware Server Console and check the inventory. To see the inverse of this operation just change the last line of the script to call the `server.UnRegisterVm` method and watch the virtual machine disappear from your inventory. It's that simple.

VmCtl

You've learned the preceding object, `VmServerCtl`, represents the physical server, so it is only appropriate that VMware provide you an object that represents a virtual machine. The `VmCtl` object does just that. The following tables go through the various properties and methods available from the `VmCtl` object, and after that you will walk through an example.

> *Remember that I am not listing the ESX-only properties and methods so this list is a little shorter than the actual API documentation.*

Properties

Name	Type	Access	Description
`ExecutionState`	VmExecution State	Get	Gets the current state of the virtual machine. This property returns the enumeration `VmExecutionState`. For a list of possible values refer to the enumeration section later in the chapter.

Name	Type	Access	Description
`PendingQuestion`	VmQuestion	Get	If the virtual machine is currently in `VmExecutionState_Stuck` this returns a `VmQuestion` object. If the virtual machine is not in a stuck state this property throws an exception. The virtual machine state can be retrieved from the `RegisteredVmNames` property. For more information refer to the `VmQuestion` object section later in this chapter.
`GuestInfo (<keyname>)`	string	Get/Set	Gets or sets a shared variable with a string identified by `<keyname>`. This is used to pass information between the guest and host.
`Config (<keyname>)`	string	Get/Set	This gets or sets the value of a configuration variable defined by `<keyname>` for the current virtual machine. When a virtual machine is powered on the contents of its configuration file are written to memory. This property allows you to modify those values while the machine is in operation. The changes are discarded when the machine is powered off.
`ConfigFileName`	string	Get	Gets the configuration file name for the current virtual machine.
`Heartbeat`	integer	Get	Gets the heartbeat count from the current virtual machine's VMware Tools service running on the guest OS. VMware Tools must be installed and running on the guest machine for this to function. The count is reset to 0 when the virtual machine is powered on and increments by one per second under normal load.
`ToolsLastActive`	integer	Get	This works in tandem with the preceding property. This returns the time in seconds since the last heartbeat was detected. See the following table for more information.

Table continued on following page

Name	Type	Access	Description
`DeviceIs Connected (<devicename>)`	Boolean	Get	Returns true if the device specified by `<devicename>` is connected.
`ProductInfo (VmProdInfoType)`	integer, VmProduct, VmPlatform	Get	Returns integer, VmProduct, or VmPlatform depending upon the type of product info you request. See the `VmProdInfoType`, `VmProduct`, and `VmPlatform` enumerations later in this chapter for the values you can request and that are returned with this property.
`Uptime`	integer	Get	Gets the uptime in seconds of the current virtual machine.
`Pid`	integer	Get	Get the process ID of the current virtual machine.

The `ToolsLastActive` property can be used to monitor the general health of your virtual machine. It maintains a count, in seconds, since the receipt of the last heartbeat. VMware provides the following information as a standard to use in determining the health of your guest using the `ToolsLastActive` property.

> *These numbers are not hard and fast rules but simply something to go by. In my experience writing scripts to monitor a guest operating system the only thing I cared about was if the property returned a value over 5 seconds. This typically means the guest OS is stuck, the VMware Tools service has stopped, or the machine has been turned off. This is a tool that will find little use in your arsenal unless you are performing some serious high-powered scripting.*

ToolsLastActive Value	Description
0	The VMware Tools service has not started or is not running.
1	The VMware Tools service is running and appears to be operating normally.
2, 3, 4, or 5	The VMware Tools service might be running, and the guest is under a heavy load.
Greater than 5	The VMware Tools service has stopped. This could be due to a system crash or the guest is shutting down or restarting.

Methods

This is where the meat is. The majority of what you will do to your virtual machines via scripting will take place with the methods exposed by this object.

These methods also introduce what VMware calls a synchronous method, which I suppose is an accurate description. Power operations, like starting and stopping your virtual machines, have a predetermined timeout of 4 minutes. When you call the method to start a virtual machine the method will not return until that machine is started. That's what makes it synchronous — it waits until the operation is done before it continues. Most other operations you perform with the API have a default timeout of 2 minutes.

Name	Description
`Connect(VmConnectParams, <vmName>)`	This method connects the parent object to the VMware Server specified in the `VmConnectParams` object and the virtual machine specified by `<vmName>`. Refer to the `VmConnectParams` object documentation for more information. The `<vmName>` parameter is a string representing the configuration file name of a registered virtual machine. The total number of connected objects cannot exceed 62 or an exception will be thrown.
`Start(VmPowerOpMode)`	This method starts the current virtual machine you are connected to or resumes the virtual machine if its state is currently suspended. `VmPowerOpMode` is an enumeration that controls how the power operation is handled. Refer to the `VmPowerOpMode` enumeration later in this chapter for more information.
`Stop(VmPowerOpMode)`	This method stops the current virtual machine you are connected to. This method will only succeed if the current virtual machine is in the `vmExecutionState_on` state. This can be retrieved by querying the `ExecutionState` property for this object. `VmPowerOpMode` is an enumeration that controls how the power operation is handled. Refer to the `VmPowerOpMode` enumeration later in this chapter for more information.
`Reset(VmPowerOpMode)`	This method stops then restarts the current virtual machine you are connected to. This method will only succeed if the current virtual machine is in the `vmExecutionState_on` state. This can be retrieved by querying the `ExecutionState` property for this object. `VmPowerOpMode` is an enumeration that controls how the power operation is handled. Refer to the `VmPowerOpMode` enumeration later in this chapter for more information.

Table continued on following page

Name	Description
`Suspend(VmPowerOpMode)`	This method suspends the current virtual machine you are connected to. This method will only succeed if the current virtual machine is in the `vmExecutionState_on` state. This can be retrieved by querying the `ExecutionState` property for this object. `VmPowerOpMode` is an enumeration that controls how the power operation is handled. Refer to the `VmPowerOpMode` enumeration later in this chapter for more information.
`AnswerQuestion(VmQuestion, <choice>)`	This method replies to a question, referenced by the `VmQuestion` object, with `<choice>` as the index of the answer. You will use this method in conjunction with the `PendingQuestion` property present in this object. If the `ExecutionState` property returns `vmExecutionState_Stuck`, then you retrieve the `VmQuestion` object from the `PendingQuestion` property and answer it with this method. Refer to the `VmQuestion` object section later in this chapter for more information.
`ConnectDevice(<devicename>)`	This method connects the device, `<devicename>`, to the current virtual machine. You could use this in combination with the `Config` property to set the configuration of a CD-ROM, for example, to use a specific ISO file. Then connect the CD-ROM drive to the virtual machine. This method will only succeed if the current virtual machine is in the `vmExecutionState_on` state. This can be retrieved by querying the `ExecutionState` property for this object.
`DisconnectDevice (<devicename>)`	This method disconnects the device, `<devicename>`, from the current virtual machine. This method will succeed only if the current virtual machine is in the `vmExecutionState_on` state. This can be retrieved by querying the `ExecutionState` property for this object.
`SetRunAsUser(<username>, <password>)`	This method sets the current virtual machine to run under the `<username>` and `<password>` specified.
`RunAsUser()`	This method simply returns the name of the user under which the current virtual machine is running.

Example

So now you can whip up a quick script using the `VmCtl` object. In the previous example you registered a virtual machine with your server, so you can now take it a step further. You know the name of the virtual

machine you want to use, `C:\VirtualMachines\dev-winxppro-sp2\dev-winxppro-sp2.vmx`, so you can write a script that uses both the `VmServerCtl` and `VmCtl` objects to connect to the server, verify the machine is registered, and power the machine on. This process takes place over four stages:

1. Create the `VmConnectParams` object and insert your authentication information.
2. Create a new `VmServerCtl` object using the `VmConnectParams` object discussed previously.
3. Verify the virtual machine exists in inventory.
4. Connect to the virtual machine using the `VmCtl` object and power it on.

This example assumes that you have a working VMware Server installation and at least one virtual machine configured and that it is registered in inventory.

To start off you need to create your WSH loader file that tells the scripting engine what library to load and where to find your script. Open your favorite editor and enter the following code:

example2.wsf

```
<job id="Example2">
  <reference object="VmCOM.VmCtl" />
  <script language="VBScript" src="example2.vbs" />
</job>
```

Save this file as `example2.wsf` and place it in a location where you will be keeping your scripts for this example.

Now you need to create the script itself. Open your favorite editor again and enter the following code. This script assumes a virtual machine exists on your server with a configuration file named `dev-winxppro-sp2.vmx` and is located within the `C:\VirtualMachines\dev-winxppro-sp2` directory.

example2.vbs

```
' First we create the VmConnectParams object to hold our authentication data
Set connect_params = CreateObject("VmCOM.VmConnectParams")

' Setup the connect_params to use the specific server and user we choose
' Remember, if the port is not specified then it uses the default of 902
connect_params.Hostname = "vmserver-01"
connect_params.Username = "administrator"
connect_params.Password = "P@ssword!"

' Here we create a new VmServerCtl object to represent our server
Set server = CreateObject("VmCOM.VmServerCtl")

' Call the Connect method and pass our authentication info set above
server.Connect connect_params

' Grab all of the registered virtual machines into a VmCollection object
' See the VmCollection object reference later in the chapter
Set VmCollection = server.RegisteredVmNames

' Set the name of the virtual machine we want to use
```

(continued)

example2.vbs *(continued)*

```
guest = "C:\VirtualMachines\dev-winxppro-sp2\dev-winxppro-sp2.vmx"

' Iterate through our collection and see if the virtual machine we want
' to use is in the list, if not add it to the inventory.
for each vmName in VmCollection
  ' Check to see if your virtual machine is in inventory
  If vmName = guest Then

    ' Using a regular expression to strip out the virtual machine path
    Set nameOnly = New RegExp
    nameOnly.Pattern = "[^\\]+$"
    Set match = nameOnly.Execute(vmName)

    Wscript.Echo(match(0) & " is registered." & VBCrLf & VBTab &  _
      "Attempting to start." & VBCrLf)
    ' Start the virtual machine.
    StartVm connect_Params, guest
    success = true
  End If
Next

If success <> true Then
  WScript.Echo("Something went wrong, make sure the virtual machine is _
    registered.")
End If

Function StartVm(connect_params, guest)
  Set vm = CreateObject("VmCOM.VmCtl")
  vm.connect connect_params, guest
  convertstate(vm)
    If vm.ExecutionState = vmExecutionState_Off or vm.ExecutionState = _
      vmExecutionState_Suspended Then
      vm.start(vmPowerOpMode_Soft)
    Else
      WScript.Echo("There was a problem.  The virtual machine you _
        are attempting to start is currently in a " & _
        ConvertState(vm) & " state and cannot continue.")
    End if
End Function

' This quick function is useful in all your scripts.
' It converts the integer value returned by VmCtl.ExecutionState to text.
Function ConvertState(vm)
  Select Case vm.ExecutionState
    Case vmExecutionState_On
      ConvertState = "ON"
    Case vmExecutionState_Off
      ConvertState = "OFF"
    Case vmExecutionState_Suspended
      ConvertState = "SUSPENDED"
    Case vmExecutionState_Stuck
      ConvertState = "STUCK"
    Case Else
      ConvertState = "UNKNOWN"
  End Select
End Function
```

Save this file as `example2.vbs` and place it in the same location as the `example2.wsf` file you created previously.

So, start it and see what happens. You start the script by opening a command prompt and typing the following at the command line within the directory you saved the scripts to:

```
cscript //nologo example2.wsf
```

If all goes well you should see output similar to the following:

```
dev-winxppro-sp2.vmx is registered.
        Attempting to start.
```

In order to verify the script did in fact start the virtual machine, open a VMware Server Console and check the pane for that virtual machine. It should in fact be starting up.

VmCollection

The last two objects are rather simple and really exist only to help support the preceding objects. The `VmCollection` object is simply a collection, or list if you will, of items from your server of virtual machine objects. There are two places in which you will find the `VmCollection` object:

- The first is with the `VmServerCtl.RegisteredVmNames` property where it provides a collection of registered virtual machines on the server in the form of a `VmCollection` object.
- The second is with the `VmQuestion.Choices` property covered in the next section of the chapter.

This object is very handy because it contains a list of items as strings and assigns an index to each, starting with index 1. This allows easy manipulation with functions like Visual Basic's `for each` statement to iterate through a collection of items. I provide an example at the end of this section where I iterate through a `VmCollection` object of `RegisteredVmNames` from a server.

Properties

Name	Type	Access	Description
`Count`	integer	Get	Gets the number of items in the collection and returns it as an integer. The count is 1-based so the first item will always be index 1.
`Item(<index>)`	string	Get	Gets the item at `<index>` and returns it as a string.

Example

You already used the `VmCollection` object in the `VmCtl` example; however, this time you will connect to the VMware Server, iterate a list of registered virtual machines, and display their state nice and neat on the screen. If there are no virtual machines registered, it will simply state that and exit. Here are the steps involved in this example:

1. Create the `VmConnectParams` object and insert your authentication information.
2. Create a new `VmServerCtl` object using the `VmConnectParams` object discussed previously.

3. Grab the registered virtual machine names, if any, and place them into a `VmCollection` object.
4. Iterate through the collection providing the name and current power state of each.

This example assumes that you have a working VMware Server installation and at least one virtual machine configured. The example works best if at least one virtual machine is registered in inventory.

To start off you need to create your WSH loader file that tells the scripting engine what library to load and where to find your script. Open your favorite editor and enter the following:

example3.wsf

```
<job id="Example3">
  <reference object="VmCOM.VmCtl" />
  <script language="VBScript" src="example3.vbs" />
</job>
```

Save this file as `example3.wsf` and place it in a location where you will be keeping your scripts for this example.

Time to create the script itself. Open your favorite editor again and enter the following:

example3.vbs

```
' First we create the VmConnectparams object to hold our authentication data
Set connect_params = CreateObject("VmCOM.VmConnectParams")

' Setup the connect_params to use the specific server and user we choose
' Remember, if the port is not specified then it uses the default of 902
connect_params.Hostname = "vmserver-01"
connect_params.Username = "administrator"
connect_params.Password = "P@ssword!"

' Here we create a new VmServerCtl object to represent our server
Set server = CreateObject("VmCOM.VmServerCtl")

' Call the Connect method and pass our authentication info set above
server.Connect connect_params

' Grab all of the registered virtual machines into a VmCollection object
' See the VmCollection object reference later in the chapter
Set VmCollection = server.RegisteredVmNames

' Iterate through our collection and see if the virtual machine we want
' to use is in the list, if not add it to the inventory.
If VmCollection.Count <> 0 Then
  WScript.Echo("Name" & VBTab & TBTab & VBTab & TBTab & VBTab & TBTab _
    & "Current State")

  For each vmName in VmCollection

    ' Using a regular expression to strip out the virtual machine path
    Set nameOnly = New RegExp
    nameOnly.Pattern = "[^\\]+$"
```

```
    Set match = nameOnly.Execute(vmName)

    ' Create the virtual machine object so we can ask its state
    Set vm = CreateObject("VmCOM.VmCtl")
    vm.connect connect_params, vmName

    WScript.Echo(match(0) & VBTab & TBTab & VBTab & TBTab & VBTab _
      & TBTab & ConvertState(vm))
  Next
Else
  WScript.Echo("There are no virtual machines registered.")
End If

' This quick function is useful in all your scripts.
' It converts the integer value returned by VmCtl.ExecutionState to text.
Function ConvertState(vm)
  Select Case vm.ExecutionState
    Case vmExecutionState_On
      ConvertState = "ON"
    Case vmExecutionState_Off
      ConvertState = "OFF"
    Case vmExecutionState_Suspended
      ConvertState = "SUSPENDED"
    Case vmExecutionState_Stuck
      ConvertState = "STUCK"
    Case Else
      ConvertState = "UNKNOWN"
  End Select
End Function
```

Save this file as `example3.vbs` and place it in the same location as the `example3.wsf` file you created previously.

So, start it and see what happens. You start the script by opening a command prompt and typing the following at the command line within the directory you saved the scripts to:

```
cscript //nologo example3.wsf
```

If all goes well, you should see output similar to the following:

```
Name                    Current State
dev-winxppro-sp2.vmx    OFF
```

If the result is blank, verify you have at least one virtual machine in inventory.

VmQuestion

This is the last object for the VmCOM API. The `VmQuestion` object serves as the return type for the `VmCtl.PendingQuestion()` property and contains a list of the questions pending for a virtual machine in a `vmExecutionState_Stuck` state. The collection also serves as an argument for the `VmCtl.AnswerQuestion()` method whereby you pass the `VmQuestion` object and the index of the answer you select back to the virtual machine. The example at the end of this section creates a scenario where a virtual machine enters the `vmExecutionState_Stuck` state and provides you with a question that needs answering.

Properties

Name	Type	Access	Description
`Text`	string	Get	Gets the question as a string.
`Choices`	VmCollection	Get	Gets the list of possible answers and returns them as a `VmCollection` object.
`Id`	integer	Get	Gets the ID used internally by the API to track the question number.

Example

Here comes the last example for VmCOM. You are going to force a `vmExecutionState_Stuck` state for the virtual machine so you can iterate through the `VmCollection` of questions and give an answer. Here are the steps involved in this example:

1. Create the `VmConnectParams` object and insert your authentication information.
2. Create a new `VmCtl` object using the `VmConnectParams` object discussed previously and the name of your virtual machine, `C:\VirtualMachines\dev-winxppro-sp2\dev-winxppro-sp2.vmx`.
3. Verify the machine is in a `VmExecutionState_Stuck` state and display the pending question with its choices.

This example assumes that you have a working VMware Server installation and at least one virtual machine configured. The virtual machine needs to already be registered in inventory.

There's one last piece of information you need in order for this example to work and that's how you force a stuck state. Of course this isn't something you would necessarily want to do normally but for the purpose of this example you need it to happen.

The quickest way to force a virtual machine into a stuck state is to first make sure it is powered down and removed from inventory via the VMware Server Console. Then, rename the directory in which it resides. Just one letter difference is enough. Once you do that, re-add the virtual machine to the console inventory and run the script that follows. You will be prompted with the answer and its choices.

Now, as usual, you need to create your WSH loader file that tells the scripting engine what library to load and where to find your script. Open your favorite editor and enter the code that follows:

example4.wsf

```
<job id="Example4">
  <reference object="VmCOM.VmCtl" />
  <script language="VBScript" src="example4.vbs" />
</job>
```

Save this file as `example4.wsf` and place it in a location where you will be keeping your scripts for this example.

Time to create the script itself. Open your favorite editor again and enter the following code:

example4.vbs

```
' First we create the VmConnectparams object to hold our authentication data
Set connect_params = CreateObject("VmCOM.VmConnectParams")

' Setup the connect_params to use the specific server and user we choose
' Remember, if the port is not specified then it uses the default of 902
connect_params.Hostname = "vmserver-01"
connect_params.Username = "administrator"
connect_params.Password = "P@ssword!"

' Here we create a new VmServerCtl object to represent our server
Set server = CreateObject("VmCOM.VmServerCtl")

' Call the Connect method and pass our authentication info set above
server.Connect connect_params

' Grab all of the registered virtual machines into a VmCollection object
' See the VmCollection object reference later in the chapter
Set VmCollection = server.RegisteredVmNames

' This is new to our scripts but allows us to capture
' the errors and handle them instead of just exiting
On Error Resume Next

' Iterate through our collection and see if the virtual machine we want
' to use is in the list, if not add it to the inventory.
If VmCollection.Count <> 0 Then
  For each vmName in VmCollection

    ' Create the virtual machine object so we can ask its state
    Set vm = CreateObject("VmCOM.VmCtl")
    vm.Connect connect_params, vmName
    vm.Start(vmPowerOpMode_Soft)

    ' Using a regular expression to strip out the virtual machine path
    Set nameOnly = New RegExp
    nameOnly.Pattern = "[^\\]+$"
    Set match = nameOnly.Execute(vmName)

    ' See if our virtual machine is in fact stuck
    If vm.ExecutionState = vmExecutionState_Stuck Then

      ' Retrieve our question and the possible answers
      Set question = vm.PendingQuestion
      Set answers = question.choices

      msg = VBCrLf & match(0) & " needs the following information. " _
        & VBCrLf & VBCrLf & question.text & VBCrLf

      For each answer in answers
        ans = ans & [answer] & VBCrLf
      Next
```

(continued)

example4.vbs *(continued)*

```
        WScript.Echo(msg)
        WScript.Echo(ans)
      Else
        WScript.Echo(match(0) & " appears to be functioning normally or _
          powered off.")
      End If
    Next
  Else
    WScript.Echo("There are no virtual machines registered.")
  End If

  ' This quick function is useful in all your scripts.
  ' It converts the integer value returned by VmCtl.ExecutionState to text.
  Function ConvertState(vm)
    Select Case vm.ExecutionState
      Case vmExecutionState_On
        ConvertState = "ON"
      Case vmExecutionState_Off
        ConvertState = "OFF"
      Case vmExecutionState_Suspended
        ConvertState = "SUSPENDED"
      Case vmExecutionState_Stuck
        ConvertState = "STUCK"
      Case Else
        ConvertState = "UNKNOWN"
    End Select
  End Function
```

Save this file as `example4.vbs` and place it in the same location as the `example4.wsf` file you created previously.

> *This script will attempt to power on every virtual machine in your inventory and check them for a* `vmExecuteState_Stuck`*. I recommend you only have the virtual machine in inventory that you applied the rename to previously during the prerequisites.*

So, start it and see what happens. You start the script by opening a command prompt and typing the following at the command line within the directory you saved the scripts to:

```
cscript //nologo example4.wsf
```

If all goes well you should see output similar to the following.

```
dev-winxppro-sp2.vmx needs the following information.

The location of this virtual machine's configuration file has changed since it
was last powered on.

If the virtual machine has been copied, you should create a new unique identifier
(UUID).  If it has been moved, you should keep its old identifier.

If you are not sure, create a new identifier.

What do you want to do?
```

```
Cancel
Create
Keep
Always Create
Always Keep
```

This section has provided you with insight into the five major objects used to manipulate the various pieces of your VMware Server environment. Without objects there's not much to "OOP," so a firm understanding of the objects available and their use will serve you in the coming chapters on integration. The remaining section deals with some supporting structure for the API and its objects.

Enumerations

I've used the phrase enumeration several times throughout the API reference so now you get into which ones VMware provides for you in the API:

- `VmExecutionState`
- `VmPowerOpMode`
- `VmProdInfoType`
- `VmProduct`
- `VmPlatform`

Enumerations are little more than a shortcut into helping you leverage the API. As an example in reality the API identifies execution states by an integer value, like `On = 1`; however, this is difficult to remember for the end user and prone to errors. In an attempt to not only make your life easier but also to increase the readability of your code they introduce a series of text shortcuts in order for you to refer to `vmExecutionState_On` instead of having to remember it's 2, or whatever it happens to be. Your code is more human-readable and less susceptible to errors.

> *The following enumeration reference draws on and adapts information from the official VMware Scripting Guide. I believe the following content will serve you well as a quick reference because the focus here in this book (as opposed to that in the documentation) is directly against VMware Server.*

VmExecutionState

The `VmExecutionState` enumeration lists the valid states for your virtual machines.

Value	Description
`vmExecutionState_On`	Indicates the virtual machine is on.
`vmExecutionState_Off`	Indicates the virtual machine is off.
`vmExecutionState_Suspended`	Indicates the virtual machine is suspended.

Table continued on following page

Value	Description
vmExecutionState_Stuck	Indicates the virtual machine requires user intervention, usually in the form of a question to be answered or an error.
vmExecutionState_Unknown	Just as it sounds, the virtual machine is in an unknown state.

The most common use for this enumeration is in verifying the VmExecutionState prior to performing an action against it that requires a specific state to operate. An example might be if you go to power off a virtual machine, you first check to see if it is in a vmExecutionState_On state before doing so. This allows for greater error control in your scripts.

VmPowerOpMode

The VmPowerOpMode enumeration lists the valid types of power mode operations that can be performed on your virtual machines. This is typically passed along with the several power cycling methods available in VmCtl like VmCtl.Start.

Value	Description
vmPowerOpMode_Soft	This mode has varying results based upon the type of power operation being performed. Each of these requires the installation of the VMware Tools package on the guest machine. **Start** — Powers on the virtual machine and, if installed, runs the start script located within the VMware Tools application. **Stop** — Attempts to shut down the guest operating system and then powers off the virtual machine. **Reset** — Attempts to shut down the guest operating system and then restarts the virtual machine. **Suspend** — Attempts to run a script inside the guest machine that safely releases any DHCP leases and then suspends the virtual machine.
vmPowerOpMode_Hard	**Start** — Starts the virtual machine without running any startup scripts in VMware Tools on the guest machine. **Stop, Reset, and Suspend** — Immediately powers off, resets, or suspends the virtual machine. No safeguards take place. It works just like pulling the plug from the wall.
vmPowerOpMode_TrySoft	Attempts a vmPowerOpMode_Soft action and, if it fails, resorts to vmPowerOpMode_Hard.

VmProdInfoType

The `VmProdInfoType` enumeration lists the types of product info you can retrieve while using the `VmServerCtl.ProductInfo` property.

Value	Description
`vmProdInfo_Product`	The VMware product type returned as a `VmProduct` enumeration. See the `VmProduct` enumeration section for the possible values.
`vmProdInfo_Platform`	The host platform type returned as a `VmPlatform` enumeration. See the `VmProduct` enumeration section for the possible values.
`vmProdInfo_Build`	The VMware Server build number.
`vmProdInfo_Version_Major`	The VMware Server major version number.
`vmProdInfo_Version_Minor`	The VMware Server minor version number.
`vmProdInfo_Version_Revision`	The VMware Server revision number.

VmProduct

The `VmProduct` enumeration lists the VMware product type that you are connected to.

Value	Description
`vmProduct_WS`	VMware Workstation
`vmProduct_GSX`	VMware GSX Server
`vmProduct_ESX`	VMware ESX Server
`vmProduct_UNKNOWN`	The VMware product type is unknown.

You will notice right off that VMware Server is not on this list, and it may never be. The VMware Scripting APIs have been officially deprecated and while they still function just fine with VMware Server and remain as your only free alternative, they will most likely never recognize VMware Server in this enumeration. Because I am focusing on free virtualization, it's only appropriate I stick with the APIs.

If you are curious, VMware Server returns as type `vmProduct_UNKNOWN`; however, this should cause you little or no trouble in your scripts.

VmPlatform

The final enumeration, `VmPlatform`, contains the list of valid platform types.

The Future of the Scripting APIs

Over the past two chapters I've hinted around at the depreciation of the scripting APIs by VMware. It is in fact true, and VMware will be pushing the Programming API (discussed in Chapter 10) as the supported scripting method in the future. This does not, however, mean the VmCOM and VmPerl APIs are useless. Although VMware will not be distributing any new versions of these APIs, they are in fact still very useful and continue to perform well with VMware Server. Until the Programming API takes a more active role in the product line and gains a stronger customer acceptance level, the Perl and COM APIs will remain as dominant scripting methods.

Value	Description
`vmPlatform_WINDOWS`	The host is running on a Windows-based system.
`vmPlatform_LINUX`	The host is running on a Linux-based system.
`vmPlatform_VMNIX`	The host is running on the ESX Server console.
`vmPlatform_UNKNOWN`	The host platform is unknown.

Summary

So that's the VmCOM API in a nutshell. Hopefully the API reference and bits of sample code have helped you get a better grasp on its functionality. The purpose here is not to convince you to use the VmCOM API over VmPerl or C but to simply give you the information needed in order for you to make that decision yourself.

In review you looked at the five different objects available:

- `VmConnectParams`
- `VmServerCtl`
- `VmCtl`
- `VmQuestion`
- `VmCollection`

Additionally, you reviewed the enumerations provided to help you write cleaner, more efficient code:

- `VmExecutionState`
- `VmPowerOpMode`
- `VmProdInfoType`

- `VmProduct`
- `VmPlatform`

That's the toolset. You create your `VmConnectParams` object, inject your credentials, and then create either a `VmServerCtl` or `VmCtl` object using those credentials, and you twist and tweak your virtual machines how you see fit.

Next you take a look at the VmPerl API and see some of the very same information you did here. Although you may see some basic repetition in the next chapter because the APIs are nearly identical, you will nevertheless see a new set of examples utilizing Perl.

9

Using VmPerl

Perl is a wonderful language. If you have no experience with it, I recommend you change that. Perl was originally released in 1987 as a free language intended to augment several features its creator, Larry Wall, liked from other tools available at the time on the Unix platform. Since then Perl has evolved into a rich, robust language, all while remaining free just as its inventor had intended. Many refer to Perl as the "duct tape of the Internet" and rightly so. From an administration tool to CGI and web content Perl has evolved into language applicable to a wide range of uses.

This chapter covers the VMware API for Perl. You will find that the VmPerl API offers the same functionality as its counterpart, VmCOM. As I mentioned in the previous chapter's introduction, what you use is up to you and your comfort level. I do still recommend, however, that you review all of the API offerings before you make that selection.

Perl is an interpreted language in that it requires a separate program, the Perl interpreter, for it to parse and execute the scripts. This differs from a language like C where the code is compiled and able to execute alone. In the VmCOM chapter, you used a tool called WSH, or the Windows Scripting Host, to execute the VBScripts. This program merely interpreted the VBScript file and executed its commands. Perl is no different.

> *This chapter is not an introduction to Perl, nor am I able to define all the terms and functions present here. The basics of object-oriented programming (OOP) I outlined in the preceding chapter still apply; however, the language itself is quite a bit different than the VBScript shown before. Dozens of good Perl books for all levels of learning are available, and I encourage you to grab one if Perl is something you want to pursue.*

Cross-Platform Perl Scripts

The scripts you write in this chapter can be run against the version of Perl that VMware kindly provided or against an installed version of Perl if one happens to be present on the machine where the scripts are executed. This is controlled by something I introduced back in Chapter 7 called the

shebang, which is just short for *hash bang*. This is simply a declaration at the beginning of every Perl script that tells it what to use to parse and execute the script file. For Linux-based systems, it will most likely be the system's installed version of Perl, typically located in `/usr/bin/perl`. This location obviously doesn't exist on a Windows system, and because most Windows machines do not have a copy of Perl natively, the script on Windows will have to be called using the `perl.exe` file located within the API directory. More on how to do that in the first example.

> *The* `C:\Program Files\VMware\VMware VmPerl Scripting API` *must be in your system path or the script must be located within the directory mentioned previously in order for it to function correctly.*

Normally a Perl script would have one declaration at the very beginning of a script like the one shown as follows:

```
#!/usr/bin/perl -w
```

This is the shebang and tells the system to use the copy of Perl installed in `/usr/bin` to execute this script. Also, you might notice the addition of a `-w` switch to the statement. I encourage you to use this switch at the very least during your development phase because it turns on debugging information from the interpreter. It is, however, not required.

What does the shebang actually do? Well, on Linux it tells the shell where to look for the binary used to parse and execute the script. On Windows it is of little importance; however, to ensure the scripts will port from one platform to another I suggest you include it anyway.

I mentioned I will be making the scripts platform friendly. Perl has a series of libraries that provide it with the functions necessary to run your scripts. On Linux the location of these libraries is known and therefore taken care of for you. On Windows, however, you need to tell it where to find the libraries that are specific for the Windows platform. You can do this by utilizing one of the many available environment variables Perl exposes, `$^O`. The `$^O` scalar, or by its longer name `$OSNAME`, returns the platform that Perl is compiled for. For higher level Perl scripting this provides little use on a Windows machine because all Windows machines simply return `MSWin32`; however, this is perfect for your purposes here because you only care if it's Windows, not what specific version of Windows. Now I already mentioned that you need to point Perl to the libraries specific to Windows. You can do this and still maintain the script's functionality for Linux at the same time, using the `$^O` scalar. The code that follows accomplishes this task for you.

> *If you are not familiar with Perl itself the term* scalar *might be new to you. A* scalar *is a simple fundamental type of data. Scalars in Perl can contain either a number or a string.*

```
#!/usr/bin/perl -w

BEGIN {
  if ($^O eq "MSWin32") {
    @INC = ('C:\Program Files\VMware\VMware VMPerl Scripting
      API\perl5\site_perl\5.005', 'C:\Program Files\VMware\VMware VMPerl Scripting
      API\perl5\site_perl\5.005\MSWin32-x86');
  }
}
```

Okay, so what in the world did this just do? Well, by checking the environment scalar `$^O` to see if it is equal to `"MSWin32"`, you can conditionally add, via the `@INC` array, the location of the Windows-specific libraries for Perl. On Linux this section evaluates to false and is skipped.

Presto, cross-platform functionality. By including this snippet in each of your Perl scripts you allow it to adjust to the platform in which it is being run. Pretty slick; however, I cannot take credit for it. This is a very common exercise and can be found in many cross-platform–aware Perl scripts. There is, however, one catch. It must be the absolute first thing included in the script after the shebang.

You can continue now by moving straight into the modules available via the VmPerl API.

> *The following reference draws on and adapts information from the official VMware Scripting Guide. I believe the following will serve you well as a quick reference because the focus here in this book (as opposed to that in the documentation) is directly against VMware Server.*

Modules

The VmPerl API consists of four major modules that, as is the case with the other APIs, represent the tangible pieces of your VMware Server environment. The modules you will be utilizing are in the following list:

- `VMware::VmPerl::ConnectParams`
- `VMware::VmPerl::Server`
- `VMware::VmPerl::VM`
- `VMware::VmPerl::Question`

Each module has a specific purpose and, as you will see in the following sections, can be leveraged through the various methods exposed.

> *The concept of properties introduced during the VmCOM section does not exist with Perl. In Perl all methods return a value and therefore the need for properties, in the terms you are already familiar with, are not necessary.*

VMware::VmPerl::ConnectParams

`VMware::VmPerl::ConnectParams` is your all-access pass necessary to connect to your server or virtual machine objects via the API. Both the `$server->connect()` and `$vm->connect()` methods you will see in a bit rely on the `ConnectParams` object for the access credentials required to connect.

The typical usage of this module is to create and provide the necessary information in one call utilizing its constructor, `new()`:

```
VMware::VmPerl::ConnectParams::new($hostname, $port, $username, $password)
```

You can also create the `ConnectParams` object, set the parameters via the methods that follow, and simply call the `new()` constructor with no arguments as well. This boils down to preference and how readable you want your code. Perl developers tend to pride themselves on brevity; however, I recommend you go for what will be the easiest to maintain over time.

Methods

Method	Description
`$connectparams->get_hostname()`	This method gets the name of the server. `$connectparams->get_hostname()` returns the host name as a scalar if it has been defined; otherwise, it returns `undef`. If the host name is not defined, authentication will be performed on the local machine.
`$connectparams->set_hostname($hostname)`	This method sets the name of the server. `$connectparams->set_hostname($hostname)` sets the server name defined by the scalar `$hostname`.
`$connectparams->get_port()`	This method gets the port number you wish to connect to the server with. `$connectparams->get_port()` returns a port number as a scalar if it has been defined; otherwise, it returns `undef`. If the port number is `undef` the default value of 902 will be used.
`$connectparams->set_port($port)`	This method sets the port number to use. `$connectparams->set_port($port)` sets the port to the number provided by the scalar `$port`.
`$connectparams->get_username()`	This method gets the user name set to use for authentication with the server. `$connectparams->get_username()` returns a scalar containing the user name or `undef` if one has not been provided.
`$connectparams->set_username($username)`	This method sets the user name you wish to authenticate with. `$connectparams->set_username($username)` sets the user name to the value provided by the scalar `$username`.
`$connectparams->get_password()`	This method gets the password associated with the user name from the preceding method. `$connectparams->get_password()`returns a scalar containing the password; otherwise, it returns `undef`.
`$connectparams->set_password($password)`	This method sets the password that corresponds to the user name in the preceding method. `$connectparams->set_password($password)` sets the password to the value provide in the scalar `$password`.

This object in and of itself provides nothing other than an "access card" used to connect to either your server or virtual machines. An example of its usage is provided later in the chapter.

VMware::VmPerl::Server

The server module is the doorway that you will go through for almost every script you write. To perform most functions on your server, you will be creating a `VMware::VmPerl::Server` object. The previous module, `VMware::VmPerl::ConnectParams`, is the key to that door.

There are two sets of methods in the following tables:

- The first contains methods that you can reference and use regardless of whether or not you are currently connected to a VMware Server.
- The second requires you to already be connected to a valid VMware Server.

I will be documenting the API functions that are relevant to only the version of VMware you are using, in this case VMware Server. The official API documentation contains more methods; however, they apply only to VMware ESX Server.

Methods

Method	Description
`$server->connect($connectparams)`	This method connects the object, referenced by `$server`, to a VMware Server utilizing the `Connect Params` object referenced by `$connectparams`. `$server->connect($connectparams)` returns a value that can equate to true if the connection is successful. Otherwise it returns `undef`.
`$server->get_last_error()`	This method returns the last error that occurred as an array with the following format: `[$error_number, $error_string]` This is commonly used to determine a failed result from the `connect()` and `is_connected()` methods.
`$server->is_connected()`	This method returns a value that can equate to true if the object referenced by `$server` is currently in a connect state. Otherwise it returns `undef`.

The next set of methods can be used only after you successfully connect to a VMware Server utilizing the `$server->connect()` method from the preceding table. A common way of ensuring you are connected prior to using one of the methods in the next table is by using the `$server->is_connected()` method. I provide an example of that shortly.

Method	Description
`$server->registered_vm_names()`	This method returns an array of strings that contains the configuration file paths and names to all the currently registered, in inventory, virtual machines on `$server`. If no virtual machines exist in inventory it returns an empty list.
`$server->register_vm($vmname)`	This method registers a new virtual machine on `$server` where `$vmname` is the name of the configuration file.
`$server->unregister_vm($vmname)`	This method unregisters a virtual machine on `$server` where `$vmname` is the name of the configuration file.

Example

So how about you crank out your first Perl example? In this script I connect to a VMware Server utilizing the `ConnectParams` and `Server` modules and attempt to register a virtual machine.

To make things simpler I recommend, at least for now, that you save your script into the `C:\Program Files\VMware\VMware VmPerl Scripting API` directory. If you would rather not do that simply add the preceding directory to your system path. You can do this by selecting My Computer➪Properties➪Advanced and finally the Environment Variables button. Scroll through the list until you find "Path" and select Edit. Remember, if you have a command-line console open you will need to close it and reopen in order for the new path to take effect.

> **For Linux users none of this is an issue. You should be able to save and run your scripts from any location because Perl and its libraries are already part of your path.**

You will start to see my examples take on a little more life. The examples I provided during the VmCOM chapter demonstrated basic functionality and left something to be desired in terms of error handling. In the Perl examples and throughout the rest of the book the code will start to take on a little more shape. This particular example adds the ability to pass the arguments in from the command line as well as some basic error handling.

This example assumes you already have a working VMware Server installation and at least one virtual machine configured but not currently in inventory.

Enter the following code using your favorite text editor and save it as `example1.pl`:

example1.pl

```
#!/usr/bin/perl -w
#
# Usage: perl example1.pl <path to config file> [<server> <user> <password>]
# The arguments in square brackets are optional.
```

```
# This is where you add the Windows specific libraries if the platform is MSWin32
BEGIN {
  if ($^O eq "MSWin32") {
     @INC = (
        'C:\Program Files\VMware\VMware VmPerl Scripting
         API\perl5\site_perl\5.005',
        'C:\Program Files\VMware\VMware VmPerl Scripting
         API\perl5\site_perl\5.005\MSWin32-x86');
  }
}

use strict;

# Import the VMware Perl modules needed
use VMware::VmPerl;
use VMware::VmPerl::Server;
use VMware::VmPerl::ConnectParams;

# Here we check the command line for arguments and if none we print out
# the proper usage.
if (@ARGV < 1) {
  print ("Usage $0: <path to config file> [<server> <user> <password>]\n\n");
  print ("Place the config file path inside double quotes if the path contains
    spaces.\n");
  exit(1);
}

# Read in our command line arguments.
my ($config_path, $server_name, $user, $password) = @ARGV;

# Set our port number.
my $port = 902;

# Create our $connectparams object to use when we connect to the server
my $connect_params = VMware::VmPerl::ConnectParams::new($server_name, $port, $user,
$password);

# Create the server object
my $server = VMware::VmPerl::Server::new();

# Thanks to Perl we can connect to our server and evaluate the success
# all in one line of code.  Here, if the connection fails we display the error.
if (!$server->connect($connect_params)) {
  my ($error_number, $error_string) = $server->get_last_error();
  die "Could not connect to server: Error $error_number: $error_string\n";
}

# Again we attempt to register the vm and if it fails some error handling takes
place.
if (!$server->register_vm($config_path)) {
  my ($error_number, $error_string) = $server->get_last_error();

  # Since the most common error with this action is a machine that already exists
  # we should handle it more gracefully.
```

(continued)

example1.pl *(continued)*

```
    if ($error_number == -20) {
      print "The virtual machine is already registered.\n\n";
      registered_vm_list();
      exit(0);
    }

    # The other common problem is that the vm cannot be found, lets handle
    # that as well.
    if ($error_number == -11) {
      print "The virutal machine could not be found.\n";
      print "Please check your path and try again.\n";
      exit(1);
    }
    die "Could not register the virtual machine: Error $error_number:
$error_string\n";

  } else {
    print ("Virtual machine registered.\n");
    registered_vm_list();
  }

  sub registered_vm_list {
    print ("The following virtual machines are available on this server.\n\n");
    print "$_\n" foreach ($server->registered_vm_names());
  }
```

To run this script simply save it under either the `C:\Program Files\VMware|VMware VmPerl Scripting API` directory or another location and add the preceding path to your system path.

You can start this with the following command and by inserting the path to the configuration file you wish to register. The items in square brackets are optional, and if left out the script will authenticate to the local machine using the current user's credentials:

```
Perl example1.pl <config file path> [<server> <username> <password>]
```

If the path to your configuration file contains spaces be sure and enclose it within double quotes on the command line.

I'm taking a few things for granted with this first example. First and foremost I'm assuming you have a working knowledge of Perl. If you don't, this may look a little cryptic to you. Everything should be pretty self-explanatory, except perhaps the way in which you evaluate the success or failure of a method. Take the following snippet as an example:

```
if (!$server->connect($connect_params)) {
  my ($error_number, $error_string) = $server->get_last_error();
  die "Could not connect to server: Error $error_number: $error_string\n";
}
```

This `if` statement simply says if you try to connect to the server and it fails, then print out the error message. The trick, if you aren't familiar with Perl, is in the `(!$server->connect($connect_params))` portion.

Remember from the preceding API reference that I pointed out that if a method failed it returned `undef`? The term `undef` is simply short for undefined. As far as Perl is concerned `undef` or 0 returned from a method equates to the Boolean value *false* and any other value returned is *true*. By prefixing the command with the `NOT` operator (!), you are saying that if NOT connect = true then perform the rest; if true then continue on with the next block of code. This allows you to evaluate the return of `undef` if the connect method fails, because Perl interprets that as false.

> *Perl doesn't actually have Boolean values or true and false built in. It evaluates any undefined or value of 0 as a false statement, whereas any other value, or text, as true. Negative 1 (-1), or "This is a string" is a true statement in Perl, whereas undef and 0 are false.*

One other thing I should point out is the use of `exit(0)` and `exit(1)` in the example. Exit, of course, just terminates the execution of the script, but the values of 0 and 1 allow you to pass the state in which it exited. Common practice is to pass zero (0) as an exit state if the script terminated normally or with an acceptable result, while one (1), or any other number other than zero, refers to an exit under other than normal circumstances. This will come in handy later when you are calling Perl scripts from your build routines and evaluating whether or not they succeeded.

VMware::VmPerl::VM

This is the bread and butter of all the modules and the one you will use in almost every single script. The VM module is what allows you to perform all the management of your individual virtual machines housed on the server. If you know the name of the virtual machine configuration file you wish to use you can even skip the server object altogether and just connect directly to the virtual machine itself.

You are about to have method overload. There are a bunch of different methods in this module so read carefully and use the right tool for the job. Like many of the VmCOM methods these are synchronous and will freeze your script until either the operation completes or it times out. The default is 2 minutes except for power operations, which are 4-minute timeouts.

The first table refers to the methods that can be used without an active connection to your server. The methods in the second table all require a preexisting connection to be made with the server before they will function correctly.

Methods

Method	Description
`$vm->connect ($connectparams, $vmname)`	This method connects the object, referenced by `$vm`, to a VMware Server and specific virtual machine utilizing the `ConnectParams` object referenced by `$connectparams` and the virtual machine configuration file name referenced by `$vmname`. `$vm->connect($connectparams, $vmname)` returns a value that can equate to true if the connection is successful. Otherwise it returns `undef`.

Table continued on following page

Method	Description
`$vm->get_last_error()`	This method returns the last error that occurred as an array with the following format. `[$error_number, $error_string]` This is commonly used to determine a failed result from the `connect()` and `is_connected()` methods.
`$vm->is_connected()`	This method returns a value that can equate to true if the object referenced by `$vm` is currently in a connect state. Otherwise it returns `undef`.

The next set of methods can only be used after you successfully connect to a VMware Server utilizing the `$vm->connect()` method from the preceding table. A common way of ensuring you are connected prior to using one of the methods in the next table is by using the `$vm->is_connected()` method. I provide an example of that shortly.

Method	Description
`$vm->start($mode)`	This method powers on a virtual machine that is currently powered off or resumes a virtual machine that was currently suspended. `$mode` refers to `VMware::VmPerl::VM_POWEROP_MODE_<xxx>`. The values for this are covered later in this chapter. If this operation fails the method will return `undef`, and the error can be retrieved via the `get_last_error()` method.
`$vm->stop($mode)`	This method shuts down the virtual machine. `$mode` refers to `VMware::VmPerl::VM_POWEROP_MODE_<xxx>`. The values for this are covered later in this chapter. This operation fails if the virtual machine is not in the `VM_EXECUTION_STATE_ON` state and will return `undef`.
`$vm->reset($mode)`	This method shuts down and reboots a virtual machine. `$mode` refers to `VMware::VmPerl::VM_POWEROP_MODE_<xxx>`. The values for this are covered later in this chapter. This operation fails if the virtual machine is not in the `VM_EXECUTION_STATE_ON` state and will return `undef`.

Method	Description
`$vm->suspend($mode)`	This method suspends a virtual machine. `$mode` refers to `VMware::VmPerl::VM_POWEROP_MODE_<xxx>`. The values for this are covered later in this chapter. This operation fails if the virtual machine is not in the `VM_EXECUTION_STATE_ON` state and will return `undef`.
`$vm->get_connected_users()`	This method returns an array of local and remotely connected users and their IP address. This includes all console and web interface users. If the operation fails, this method will return `undef`.
`$vm->get_execution_state()`	This method returns the virtual machine's current state as defined by the `VM_EXECUTION_STATE_<xxx>` values covered later in the chapter. If the operation fails, this method will return `undef`.
`$vm->get_guest_info ($keyname)`	This method is used to access a shared variable, denoted by `$keyname`, that can be accessed by both the host and guest. If the operation fails, this method will return `undef`.
`$vm->set_guest_info ($keyname, $value)`	This method sets the shared variable, `$keyname`, as `$value`. This can be used to share information between your host and guest. If the operation fails, this method will return `undef`.
`$vm->get_config_ file_name()`	This method returns a scalar containing a string that represents the configuration file name of the virtual machine you are connected to. If the operation fails, this method will return `undef`.
`$vm->get_config($keyname)`	This method gets the configuration value denoted by `$keyname` from the virtual machine's configuration file. If the operation fails, this method will return `undef`.
`$vm->set_config ($keyname, $value)`	This method sets the configuration value for the key, `$keyname`, as the value, `$value`. If the operation fails, this method will return `undef`.

Table continued on following page

Method	Description
`$vm->get_product_info ($infotype)`	This method returns the product information requested by `$infotype`. See the discussion on `$infotype` values later in the chapter. If the operation fails, this method will return `undef`.
`$vm->get_heartbeat()`	The method returns the heartbeat count generated by the VMware Tools application installed on the guest operating system. The count begins at 0 when the virtual machine is powered on and typically increments at least once per second under normal load. If the operation fails, this method will return `undef`.
`$vm->get_tools_ last_active()`	This method returns an integer indicating how much time, in seconds, has passed since the last heartbeat was detected. See the following table for more information. If the operation fails, this method will return `undef`.
`$vm->get_pending_ question()`	This method returns a `VMware::VmPerl::Question` object if the virtual machine is currently in the `VM_EXECUTION_ STATE_STUCK` state. For more information, see the discussion concerning the `Question` object later in this chapter. If the operation fails, this method will return `undef`.
`$vm->answer_question ($question, $choice)`	This method is used to reply to a question retrieved by the preceding method. `$question` refers to the `Question` object returned from the `get_pending_question` method and `$choice` represents the index of the answer you wish to choose. If the operation fails, this method will return `undef`.
`$vm->device_is_ connected($devicename)`	This method is used to determine the connection status of the device `$devicename`. If the operation fails, this method will return `undef`. This refers to virtual devices like CD-ROM drives. You can poll the connected status of such devices with this method.
`$vm->connect_device ($devicename)`	This method is used to connect the device `$devicename` to the current virtual machine. If the virtual machine is not powered on, this will result in an error. If the operation fails, this method will return `undef`. You can connect a previously configured piece of virtual hardware with this method — for example, a previously offline CD-ROM drive.

Method	Description
`$vm->disconnect_device ($devicename)`	This method disconnects the device `$devicename` from the current virtual machine. If the virtual machine is not powered on this will result in an error. If the operation fails, this method will return `undef`. You can disconnect an online piece of virtual hardware with this method.
`$vm->set_runas_user ($user, $password)`	This method allows you to set the user, `$user`, and password, `$password`, of the account you want the virtual machine to run under. If the operation fails, this method will return `undef`.
`$vm->get_runas_user()`	This method simply returns the name of the user running the virtual machine as a scalar that contains a string.

The `get_tools_last_active()` method can be used to monitor the general health of your virtual machine. It maintains a count, in seconds, since the receipt of the last heartbeat. VMware provides the following information as a standard to use in determining the health of your guest using the `get_tools_last_active()` method.

As I pointed out in the VmCOM section in the last chapter concerning this subject, these numbers are not hard and fast rules but simply something to go by.

get_tools_last_active Value	Description
0	The VMware Tools service has not started or is not running.
1	The VMware Tools service is running and appears to be operating normally.
2, 3, 4, or 5	The VMware Tools service might be running, and the guest is under a heavy load.
Greater than 5	The VMware Tools service has stopped. This could be due to a system crash or the guest is shutting down or restarting.

Example

With the next example, you start to get fancy. Here you want to script the ability to start a virtual machine, but instead of providing the name of the configuration file to start, the script will allow you to choose from a list of registered virtual machines already on the server. You will only be able to start a virtual machine that has a current state of off or suspended.

example2.pl

```
#!/usr/bin/perl -w

# This is where you add the Windows specific libraries if the platform is MSWin32
BEGIN {
  if ($^O eq "MSWin32") {
     @INC = (
        'C:\Program Files\VMware\VMware VmPerl Scripting
API\perl5\site_perl\5.005',
        'C:\Program Files\VMware\VMware VmPerl Scripting
API\perl5\site_perl\5.005\MSWin32-x86');
  }
}

use strict;

# Import the VMware Perl modules needed
use VMware::VmPerl;
use VMware::VmPerl::Server;
use VMware::VmPerl::VM;
use VMware::VmPerl::ConnectParams;

my $server_name;
my $port;
my $username;
my $password;

# This uses the built in Perl function GetOptions to retrieve the command line
# switches and stores them in their respective scalars.
sub get_args() {
  use Getopt::Long;
  my $help;
  GetOptions ("help|h|?" , \$help,
              "server|s=s" , \$server_name,
              "port|tcp=i" , \$port,
              "user|u=s" , \$username,
              "pass|p=s" , \$password);

  usage() if ($help);
}

# This is our formatted usage statement printed if the -h switch is passed.
sub usage() {
  print <<eof;

  Usage:\n $0 [-help] [-server server -tcp port\# -user username -pass password]

    Optional
    -help   : this message
    -server : The server name you wish to connect to.
    -tcp    : The port to connect through.
    -user   : The user name you wish to authenticate with.
    -pass   : The password for the user name provided by -u.
```

```
eof
exit;
}

get_args();

# This extracts the integer value for constants out of the API
# Use this as a quick helper method to perform integer to constant string
# conversions
# This is adapted from one of the VMware samples and I use it in almost every
# script.
sub constant_to_string {
  my $constant_string = shift;
  return VMware::VmPerl::constant($constant_string, 0);
}

my $state_string_map = {};
my @state_strings = ("VM_EXECUTION_STATE_ON",
                     "VM_EXECUTION_STATE_OFF",
                     "VM_EXECUTION_STATE_SUSPENDED",
                     "VM_EXECUTION_STATE_STUCK",
                     "VM_EXECUTION_STATE_UNKNOWN");

# Loop through the strings above and pull out the integer value for each from the
# API
foreach my $state_string (@state_strings) {
  $state_string_map->{constant_to_string($state_string)} = $state_string;
}

######################################################################
# Now the fancy stuff is out of the way let's work with our server.

# Create our $connectparams object to use when we connect to the server
my $connect_params =
  VMware::VmPerl::ConnectParams::new($server_name,$port,$username,$password);

# Create the server object
my $server = VMware::VmPerl::Server::new();

# Connect to the server or display the error on failure
if (!$server->connect($connect_params)) {
  my ($error_number, $error_string) = $server->get_last_error();
  die "Could not connect to server: Error $error_number: $error_string\n";
}

# Let's display a list of the available virtual machines in inventory and prompt
# for a choice to start
my @vm_list = $server->registered_vm_names();

my $number =1;
print "Choose from one of the following virtual machines.\n";

# Make sure we actually have a list to use.
if (!defined($vm_list[0])) {
  my ($error_number, $error_string) = $server->get_last_error();
```

(continued)

example2.pl *(continued)*

```
    die "Could not get a list of virtual machines from the server or no virtual
      machines exists: Error $error_number : $error_string\n";
}

# Display our registered machines and assign them a number.
foreach (@vm_list) {
  print "$number - $_\n";
  $number++;
}

print "\nEnter a choice: ";

my $choice = 0;
while ($choice < 1 or $choice > $number - 1) {
  chomp($choice = <STDIN>);
  if ($choice <1 or $choice > $number - 1) {
    print "Invalid choice.  Enter again: ";
  }
}

# Now, we know the number they picked and that the number is one more than the
# index in @vm_list. Take the number, pull the config file in question and connect
# to that vm.

# Create a new vm object
my $vm = VMware::VmPerl::VM::new();

# Connect to the vm selected above
if (!$vm->connect($connect_params, $vm_list[$choice - 1])) {
  my ($error_number, $error_string) = $vm->get_last_error();
  die "Could not connect to the virtual machine: Error $error_number :
    $error_string\n";
}

# Check the vm's execution state.  If off or suspended allow the power on.
my $power_state = $vm->get_execution_state();

if (!defined($power_state)) {
  my ($error_number, $error_string) = $vm->get_last_error();
  die "Could not get the virtual machine's power state: Error $error_number :
    $error_string\n";
} elsif ($power_state == VM_EXECUTION_STATE_OFF || $power_state ==
    VM_EXECUTION_STATE_SUSPENDED) {
  print "Powering on virutal machine\n";

  if (!$vm->start()) {
    my ($error_number, $error_string) = $vm->get_last_error();
    die "Could not power on the virtual machine: Error $error_number :
      $error_string\n";
  } else {
    print "$vm_list[$choice - 1] is now in the $state_string_map->
      {$vm->get_execution_state()} state.\n";
  }
```

```
} else {
  print "The virtual machine is in the $state_string_map->{$vm-
>get_execution_state()} state and cannot be powered on.\n";
}
```

Okay, so I got a little carried away with this one. I introduced the ability to accept command-line options, convert constants to strings, accept a choice from standard input, and a few other things. I did this to show you the sheer power available to you. This script can be modified to perform a multitude of other functions, such as offering the ability to not only power on but also power off or reset with little effort. Though it may take you a little longer to work through this example, it will certainly leave you with little doubt of just what can be accomplished.

To run this script simply save it either under the `C:\Program Files\VMware|VMware VmPerl Scripting API` directory or another location and add the preceding path to your system path. You can kick this off with the following command. The items in square brackets are optional, and if left out, the script will authenticate to the local machine using the current user's credentials:

```
Perl example2.pl -server <server name> -port <port> -user <username> -pass
<password>
```

Or if you need some command-line assistance with the syntax, issue the following command:

```
Perl example2.pl -help
```

VMware::VmPerl::Question

The final object is simple. The `Question` object exposes only three methods and is typically returned by the `VMware::VmPerl::VM::get_pending_question()` method. The only time you will really find yourself using this object is if you write any scripts that display and handle pending questions during a `VM_EXECUTION_STATE_STUCK` state.

Methods

Method	Description
`$question->get_text()`	This method returns a scalar containing the pending question. If the operation fails, this method will return `undef`.
`$question->get_choices()`	This method returns an array that contains a list of the possible answers to the question. If the operation fails, this method will return `undef`.
`$question->get_id()`	This method returns an integer that is used internally and has little use to you. If the operation fails, this method will return `undef`.

Example

You may remember this example from the VmCOM chapter. This is the one where you actually have to force the machine into a `STUCK` state in order to see this script function. The easiest way to do this is remove the virtual machine from inventory, rename its directory, add it back into inventory, and power it on. This will present you with a pending question in the VMware Server Console. While that question is present on the screen, run the script, which will display the question and allow you to answer it.

I've adapted the preceding example from this chapter and some of its sweetness into this one. Basically what it is doing is grabbing a list of the registered virtual machines from the server and one by one connecting to them and polling for a `VM_EXECUTION_STATE_STUCK` state. Once one is found it jumps into the `handle_question()` subroutine and poses the question. If you position your windows you can actually see an answer given to the script affect the question present on the console.

Enter the following into a new script named `example3.pl` with your favorite text editor:

example3.pl

```
#!/usr/bin/perl -w
#

# This is where you add the Windows specific libraries if the platform is MSWin32
BEGIN {
  if ($^O eq "MSWin32") {
    @INC = (
      'C:\Program Files\VMware\VMware VmPerl Scripting API\perl5\site_perl\5.005',
      'C:\Program Files\VMware\VMware VmPerl Scripting
        API\perl5\site_perl\5.005\MSWin32-x86');
  }
}

use strict;

# Import the VMware Perl modules needed
use VMware::VmPerl;
use VMware::VmPerl::Server;
use VMware::VmPerl::VM;
use VMware::VmPerl::ConnectParams;
use VMware::VmPerl::Question;

my $server_name;
my $port;
my $username;
my $password;

# This uses the built in Perl function GetOptions to retrieve the command line
# switches and stores them in their respective scalars.
sub get_args() {
  use Getopt::Long;
  my $help;
  GetOptions ("help|h|?" , \$help,
              "server|s=s" , \$server_name,
              "port|tcp=i" , \$port,
```

```
              "user|u=s" , \$username,
              "pass|p=s" , \$password);

  usage() if ($help);
}

# This is our formatted usage statement printed if the -h switch is passed.
sub usage() {
  print <<eof;

  Usage:\n $0 [-help] [-server server -tcp port\# -user username -pass password]

    Optional
    -help   : this message
    -server : The server name you wish to connect to.
    -tcp    : The port to connect through.
    -user   : The user name you wish to authenticate with.
    -pass   : The password for the user name provided by -u.

eof
exit;
}

get_args();

# This extracts the integer value for constants out of the API
# Use this as a quick helper method to perform integer to constant string
# conversions
# This is adapted from one of the VMware samples and I use it in almost every
# script.
sub constant_to_string {
  my $constant_string = shift;
  return VMware::VmPerl::constant($constant_string, 0);
}

my $state_string_map = {};
my @state_strings = ("VM_EXECUTION_STATE_ON",
                     "VM_EXECUTION_STATE_OFF",
                     "VM_EXECUTION_STATE_SUSPENDED",
                     "VM_EXECUTION_STATE_STUCK",
                     "VM_EXECUTION_STATE_UNKNOWN");

# Loop through the strings above and pull out the integer value for each from the
# API.
foreach my $state_string (@state_strings) {
  $state_string_map->{constant_to_string($state_string)} = $state_string;
}

######################################################################
# Now the fancy stuff is out of the way let's work with our server.

# Create our $connectparams object to use when we connect to the server
my $connect_params =
VMware::VmPerl::ConnectParams::new($server_name,$port,$username,$password);
```

(continued)

example3.pl *(continued)*

```
# Create the server object
my $server = VMware::VmPerl::Server::new();

# Connect to the server or display the error on failure
if (!$server->connect($connect_params)) {
  my ($error_number, $error_string) = $server->get_last_error();
  die "Could not connect to server: Error $error_number: $error_string\n";
}

# Let's retrieve a list of the available virtual machines in inventory
my @vm_list = $server->registered_vm_names();


# Make sure we actually have a list to use.
if (!defined($vm_list[0])) {
  my ($error_number, $error_string) = $server->get_last_error();
  die "Could not get a list of virtual machines from the server or no virtual
    machines exist in inventory: Error $error_number : $error_string\n";
}

# Now, we have a list of the registered virtual machines, let's step through each
# and check their state
my $vm = VMware::VmPerl::VM::new();

foreach my $vm_selected (@vm_list) {

  # Connect to the vm
  if (!$vm->connect($connect_params, $vm_selected)) {
    my ($error_number, $error_string) = $vm->get_last_error();
    die "Could not connect to the virtual machine: Error $error_number :
      $error_string\n";
  }

  if ($state_string_map->{$vm->get_execution_state()} eq
    "VM_EXECUTION_STATE_STUCK") {
    print "$state_string_map->{$vm->get_execution_state()}\n";
    handle_questions($vm);
  } else {
    # Print it's state
    print "No virtual machines require user intervention.\n";
  }
  undef $vm;
}

sub handle_questions {
  $vm = shift;
  print $vm->get_config_file_name() . " needs user intervention\n\n";

  my $question = $vm->get_pending_question();

  if (!defined($question)) {
    undef $vm;
    die "Could not get the pending question.\n";
  }
```

```
    my $text = $question->get_text();

    if (!defined($text)) {
      undef $vm;
      die "Could not get the question text.\n";
  }

    my @answers = $question->get_choices();

    if (!defined($answers[0])) {
      undef $vm;
      die "Could not get the answers.\n";
    }

    print "-----------------------------------------\n";
    print "\n$text\n";

    my $selected_answer = 0;

    print "Select from one of the choices below.\n";

    for (my $i =0; $i <= $#answers; $i++) {
      print "\t" . ($i + 1) . " - $answers[$i]\n";
    }

    print "\nEnter choice:";
    while ($selected_answer < 1 or $selected_answer > $#answers) {
      chomp($selected_answer = <STDIN>);
      if ($selected_answer < 1 or $selected_answer > $#answers) {
        print "Invalid choice.  Enter again: ";
      }
    }

    if (!$vm->answer_question($question, $selected_answer - 1)) {
      my ($error_number, $error_string) = $vm->get_last_error();
      die "Could not answer the question. Error $error_number : $error_string\n";
    }
  }
  undef $server;
```

To run this script simply save it under either the `C:\Program Files\VMware|VMware VmPerl Scripting API` directory or another location and add the preceding path to your system path. You can kick this off with the following command. The items in square brackets are optional, and if left out, the script will authenticate to the local machine using the current user's credentials:

```
Perl example3.pl -server <server name> -port <port> -user <username> -pass
<password>
```

Or if you need some command-line assistance with the syntax, issue the following command:

```
Perl example3.pl -help
```

Remember, for this to work you need to force a machine into a `STUCK` state or else the script will just exit because there's nothing to do.

This section introduced you to the four major modules you will leverage with the VmPerl API. These modules and their respective methods will be center stage for all your VmPerl scripts. The remaining sections serve only to support the modules and their functionality. Next, I cover the constants defined in the API.

Constants

Everything has a different name. In VmCOM you knew these values as enumerations. In Perl they are simply constants. Technically they were just constants in COM as well but that's beside the point. This section outlines the five constants exposed by the VmPerl API. As you read through the methods of each library earlier in the chapter, you no doubt noticed their use as arguments. This should give you a quick reference to the values available and what they mean.

VM_EXECUTION_STATE

The `VM_EXECUTION_STATE` constants are used to determine the operational status of your virtual machine. If you were to query your virtual machine with the `$vm->get_execution_state()` method, it would return one of the five constants listed in the following table.

Value	Description
`VM_EXECUTION_STATE_ON`	The virtual machine is powered on.
`VM_EXECUTION_STATE_OFF`	The virtual machine is powered off.
`VM_EXECUTION_STATE_SUSPENDED`	The virtual machine is in a suspended state.
`VM_EXECUTION_STATE_STUCK`	The virtual machine is stuck and requires user intervention.
`VM_EXECUTION_STATE_UNKNOWN`	The virtual machine is in an unknown state.

VM_POWEROP_MODE

The `VM_POWEROP_MODE` constants define how your virtual machine behaves during a power transition. The `$vm->start()`, `$vm->stop()`, `$vm->suspend()`, and `$vm->reset()` methods all allow you to specify one of these constants to control how the operation is handled. If no mode is specified, the power operation default is soft.

Value	Description
`VM_POWEROP_MODE_SOFT`	This mode has varying results based upon the type of power operation being performed. Each of these requires the installation of the VMware Tools package on the guest machine. **Start**—Powers on the virtual machine and, if installed, runs the startup script located within the VMware Tools application.

Value	Description
	Stop — Attempts to shut down the guest operating system and then powers off the virtual machine. **Reset** — Attempts to shut down the guest operating system and then restarts the virtual machine. **Suspend** — Attempts to run a script inside the guest machine that safely releases any DHCP leases and then suspends the virtual machine.
VM_POWEROP_MODE_HARD	**Start** — Starts the virtual machine without running any startup scripts in VMware Tools on the guest machine. **Stop, Reset, and Suspend** — Immediately powers off, resets, or suspends the virtual machine. No safeguards take place. It works just like pulling the plug from the wall.
VM_POWEROP_MODE_TRYSOFT	Attempts a VM_POWEROP_MODE_SOFT action and, if it fails, resorts to VM_POWEROP_MODE_HARD.

Infotype

If you remember from the VM module explanation earlier in the chapter, this set of constants is used with the `$vm->get_product_info()` method. The `infotype` chosen results in `$vm->get_product_info()` returning different data types.

Value	Description
VM_PRODINFO_PRODUCT	This `infotype` will cause the `get_product_info()` method to return a constant from the VM_PRODINFO_PRODUCT_<xxx> group. See the next section for more information on this constant.
VM_PRODINFO_PLATFORM	This `infotype` will cause the `get_product_info()` method to return a constant from the VM_PRODINFO_PLATFORM_<xxx> group. See the next section for more information on this constant.
VM_PRODINFO_BUILD	This `infotype` will cause the `get_product_info()` method to return a scalar containing the product's build number as an integer.
VM_PRODINFO_VERSION_MAJOR	This `infotype` will cause the `get_product_info()` method to return a scalar containing the product's major version number as an integer.

Table continued on following page

Value	Description
VM_PRODINFO_VERSION_MINOR	This infotype will cause the get_product_info() method to return a scalar containing the product's minor version number as an integer.
VM_PRODINFO_VERSION_REVISION	This infotype will cause the get_product_info() method to return a scalar containing the product's revision number as an integer.

VM_PRODINFO_PRODUCT

This set of constants is returned when you call the $vm->get_product_info(VM_PRODINFO_PRODUCT) method.

Value	Description
VM_PRODUCT_WS	The product type is VMware Workstation.
VM_PRODUCT_GSX	The product type is VMware GSX Server.
VM_PRODUCT_ESX	The product type is VMware ESX Server.
VM_PRODUCT_UNKNOWN	The product type is unknown.

The VMware Scripting APIs have been officially deprecated, and though they still function just fine with VMware Server and remain as your only free alternative, they will most likely never recognize VMware Server as a constant. That being the case VMware Server when queried with $vm->get_product_info(VM_PRODINFO_PRODUCT) returns VM_PRODUCT_UNKNOWN. This should cause you little to no trouble in your scripts, but nonetheless it is something to be aware of.

VM_PRODINFO_PLATFORM

This set of constants is returned when you call the $vm->get_product_info(VM_PRODINFO_PLATFORM) method.

Value	Description
VM_PLATFORM_WINDOWS	The platform is Microsoft Windows.
VM_PLATFORM_LINUX	The platform is Linux.
VM_PLATFORM_VMNIX	The platform is the service console for VMware ESX Server.
VM_PLATFORM_UNKNOWN	The platform is unknown.

The Future of the Scripting APIs

Over the past two chapters I've hinted around at the depreciation of the scripting APIs by VMware. It is in fact true, and VMware will be pushing the Programming API (discussed in Chapter 10) as the supported scripting method in the future. This does not, however, mean the VmCOM and VmPerl APIs are useless. Although VMware will not be distributing any new versions of these APIs, they are in fact still very useful and continue to perform well with VMware Server. Until the Programming API takes a more active role in the product line and gains a stronger customer acceptance level, the Perl and COM APIs will remain as dominant scripting methods.

Summary

Well, that's VmPerl. My hope is that through the API reference guide and provided examples you have a better understanding of its functionality and usefulness. But my standard disclaimer still applies — use the language you are most comfortable with.

In review, the following modules and their methods were covered in this chapter:

- `VMware::VmPerl::ConnectParams`
- `VMware::VmPerl::Server`
- `VMware::VmPerl::VM`
- `VMware::VmPerl::Question`

This chapter also covered the following constants, which are timesavers:

- `VM_EXECUTION_STATE`
- `VM_POWEROP_MODE`
- `infotype`
- `VM_PRODINFO_PRODUCT`
- `VM_PRODINFO_PLATFORM`

That's it; there's really little more to it. The key is to experiment and implement. The workflow is just like VmCOM — create your connection parameters, create a server and virtual machine object, and connect to each. The VmCOM examples were simpler but were designed to prove the concepts; the examples here for Perl were deliberately more complicated. The goal wasn't to confuse but to excite. The options are nearly endless.

Next you move into the final API, the Programming API. In the next chapter I dive into the newest API offering from VMware and go over its functionality and what it has to offer in the VMware Server scripting world.

10

Using the Programming API

Brand new with the VMware Server release comes the Programming API, also known as Vix. The Vix API offers a decent amount of additional functionality over its older counterparts. Limiting it is the fact that you can only develop against it in C; this will make it difficult to adopt in some shops. The Vix API currently supports only VMware Server. This means you will not be able to leverage any existing virtual machines not residing on a VMware Server host in your shop with this API. Later releases of Vix may lift this limitation and open up to supporting more VMware platforms.

Over the course of this chapter I introduce the first version of this new API and provide a few basic samples of its use.

Because this API is so new and in its first-generation release, I decided not to provide any examples utilizing Vix in Chapters 11 and 12, the integration scenarios. I wanted to provide you with real-world examples that utilize the mature, time-tested APIs. The fact that you can compile your scripts that leverage the new Vix API into executables, as well as utilize some of the new methods exposed like `VixVM_CopyFileFromHostToGuest`, make it a very favorable solution and one to watch as it develops.

> *Because this API is so new, I highly advise you to review the release notes for the Programming API if you have trouble. The release notes for version 1.0, which I use here, are available at* `http://www.vmware.com/support/developer/prog-api/ProgAPI-1.0-Release-Notes.html`.

Compiling Against Vix with Visual C++

The Vix API is made up of several header files and libraries. If you have VMware Server installed on your development machine, the necessary files are already present. If you do not, which is the more likely scenario, then you need the files provided in the following list:

- `vix.h`
- `vm_basic_types.h`
- `vix.lib`
- `vix.dll`
- `ssleay32.dll`
- `libeay32.dll`

By default on the host machine these files live in the `C:\Program Files\VMware\Vmware VIX` directory. Reach over and grab them from the host now and place them anywhere on your development machine.

> *You can also download the Programming API package as part of the Windows or Linux client distribution packages on the VMware web site at* `www.vmware.com`*. Downloading and installing the package will forego the need to copy the files over.*

Now that that is done, I'll move into what you need to do with these files. You need to do three basic things:

- First of all, the C code you write against this API must have the following `include` statement in its declaration section:

```
#include "vix.h"
```

- Second, for the compiler to locate the headers at compile time, you will have to point it to the location of the header files you include.
- Third, you will have to point the compiler to the libraries it needs as well.

Setting Up the Project

To clarify, I think it's best to just show you how to set it up step by step. I'm using Visual Studio 2005 for the samples I provide here:

1. Start by either copying the entire `C:\Program Files\VMware\VMware Vix` directory from the host server or download and install it from the Windows client package on the VMware web site. If you copy the files by hand, drop them into its matching directory on your development machine, `C:\Program Files\VMware\VMware VIX`.
2. Now, if you install Vix via the installer from VMware, it automatically adds itself to your system path. If you move the files by hand you need to perform this step manually. Assuming you placed the copied `VMware VIX` directory under `C:\Program Files\VMware`, perform the following steps:
    a. Right-click My Computer on your development machine and choose Properties. From there, select the Advanced tab shown in Figure 10-1.
    b. Click the Environment Variables button near the bottom of the tab to reveal Figure 10-2.

Figure 10-1

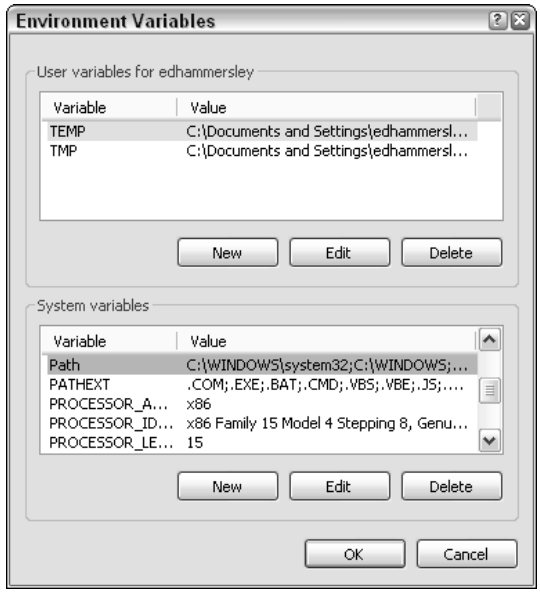

Figure 10-2

c. The lower pane is where your interest lies. Scroll through the list under System variables until you find PATH. Once there double-click it to reveal the change path window shown in Figure 10-3.

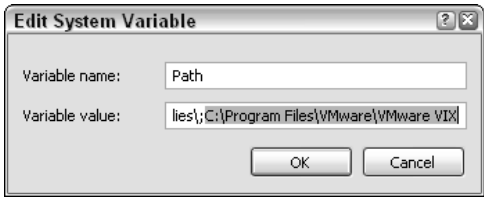

Figure 10-3

d. You may notice that I already have the path entered in the preceding figure. Add a semicolon to the end of the current path and add `C:\Program Files\VMware\VMware Vix` or the path where you stored the Vix files. Once it looks like Figure 10-3, click OK on all the dialogs. The Vix libraries are now in your system path.

3. Next you need to set up a new project in Visual Studio.

It is possible to compile your C applications from the command line much like you would in Linux with GCC; however, I'm a visual person so Visual Studio is my choice.

Open Visual Studio 2005 and select Create new Project from either the Start Page, if you have it configured to display, or the File➪New➪Project menu item shown in Figure 10-4.

Depending upon the version of Visual Studio 2005 you have installed a new project may be created simply by selecting File➪New Project.

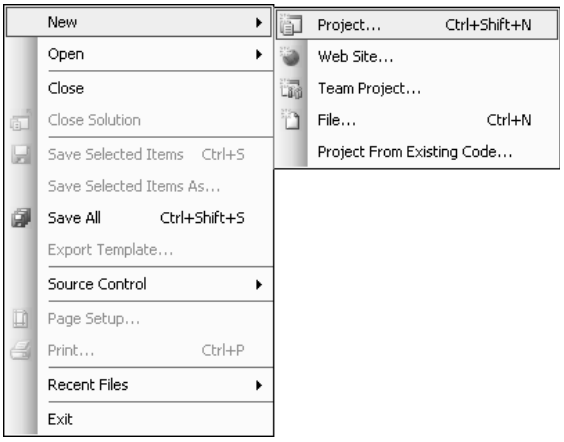

Figure 10-4

4. This will reveal Figure 10-5. There are many items in this dialog. The type of project that is at the very top of the Project Type tree will conform to what you have your development environment set for. In the figure you'll see my default type is C#. That is, however, not what you need for this project. Expand the Other Languages node and click the Visual C++ project type. This will show a gigantic list of available project types in the right-hand pane.

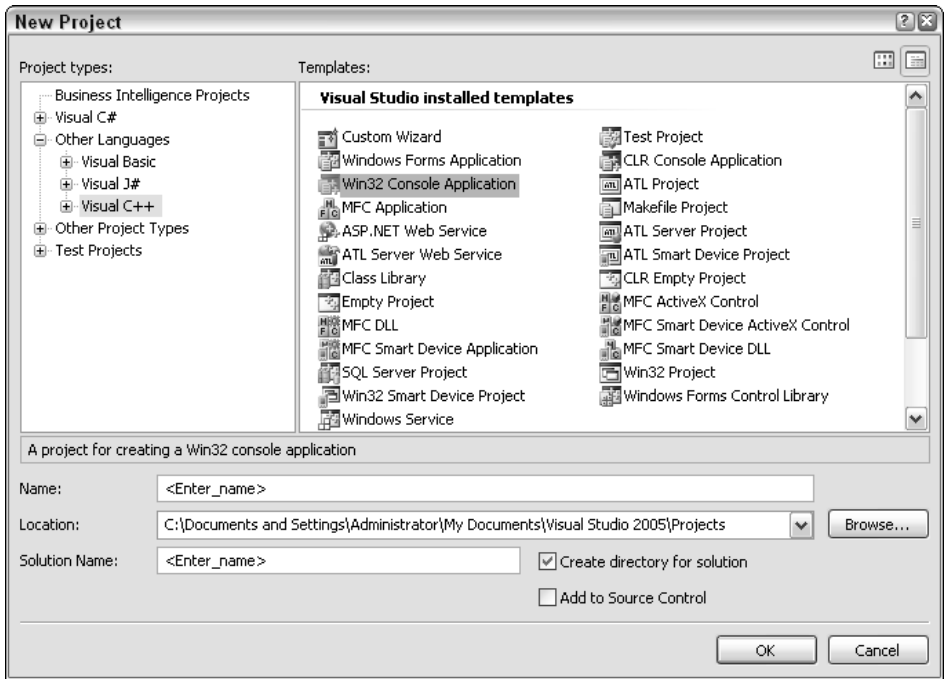

Figure 10-5

Out of the list of available project templates you are interested in the Win32 Console Application. Give it a single left click and enter the name of the project in the Name field in the lower portion of the window — name this first one as *VixSample1*. The solution name will change automatically to match the project name you gave it. Make sure Create directory for solution is checked (it should be by default) and click the OK button.

5. Once you click OK, you will be presented with the Win32 Application Wizard. This wizard is a short two-screen step-through of some basic configuration items for your project. Figure 10-6 is what you should be looking at right now.

6. Go ahead and click Next. You should see Figure 10-7. This is where you configure some specifics about your console application. All the defaults are fine. You want a console application, a pre-compiled header, and no common header files added.

Win32 Application Wizard - VixSample1
Welcome to the Win32 Application Wizard
Overview
Application Settings
These are the current project settings:
Console application
Click Finish from any window to accept the current settings.
After you create the project, see the project's readme.txt file for information about the project features and files that are generated.
< Previous
Next >
Finish
Cancel

Figure 10-6

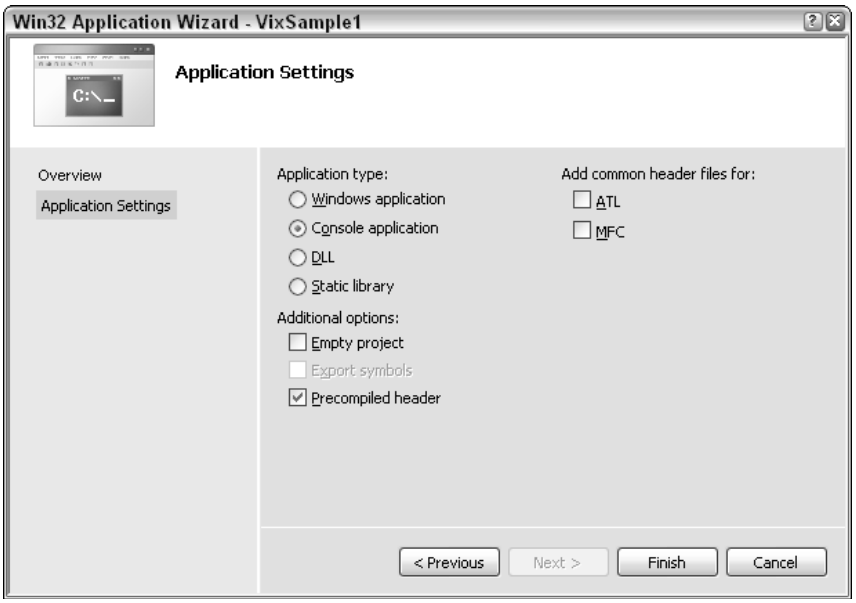

Figure 10-7

7. If your settings match the ones in the preceding figure, click Finish. After a few seconds Visual Studio will present you with the project all ready to go. The default code file the project wizard creates is called `VixSample1.cpp`, and Visual Studio should display it automatically in the editor pane. Figure 10-8 gives you a rough example of what you should see.

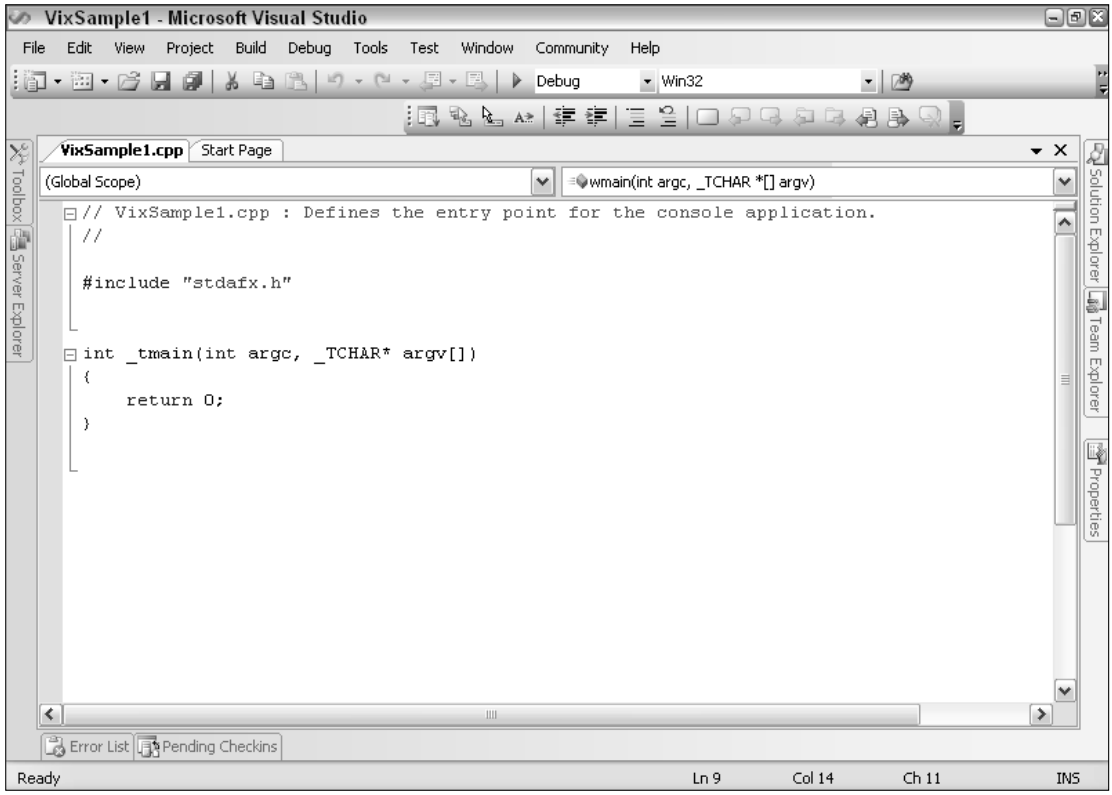

Figure 10-8

Including the include Statement

Now that the project is ready to go, you'll remember there are three things you need to address to leverage the Vix API: the include statement, the location of the header files, and the location of any libraries.

The first is your `include` statement. In the *VixSample1.cpp* editor window you have on screen enter the following `include` statement directly below the existing include statement the wizard placed in the code file:

```
#include "vix.h"
```

This `include` statement allows you to reference the functions contained with the Vix header file.

Pointing the Project to the Header Files

The `include` statement alone, however, is not enough. You need to set the project properties to look in the `C:\Program Files\VMware\VMware VIX` directory for the header file:

1. Click the Project menu and select VixSample1 Properties from the menu. This should show the window in Figure 10-9.

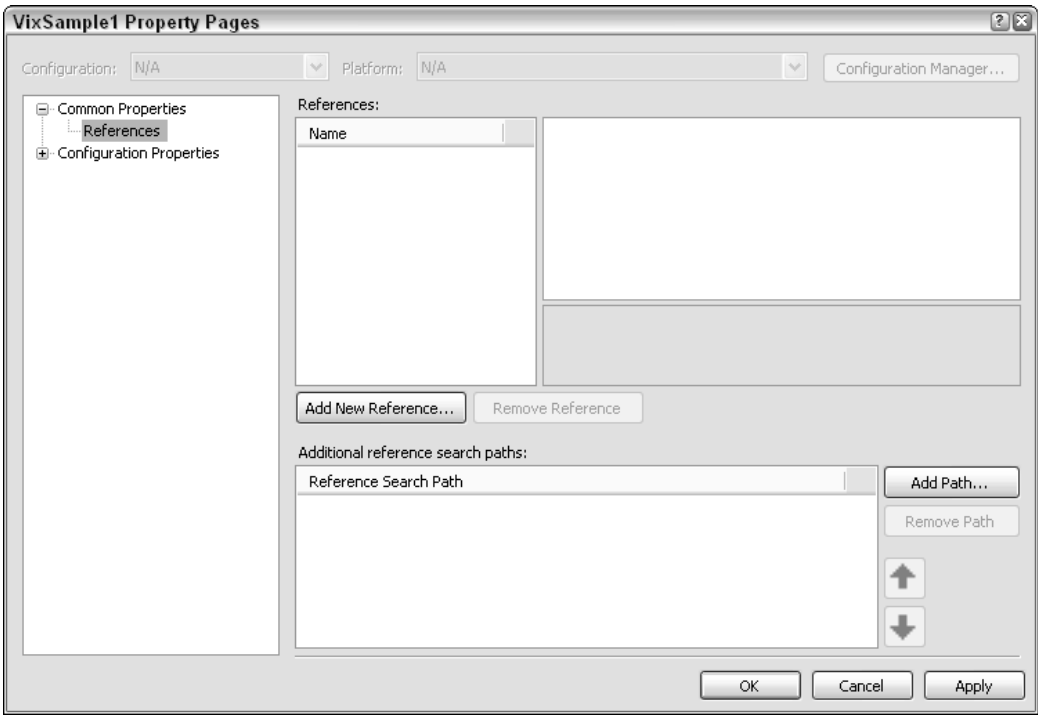

Figure 10-9

2. From here you need to expand the Configuration Properties node and select C/C++. This reveals a whole heap of stuff. Figure 10-10 puts you on track with where I'm going.
3. If you look closely when you have the C/C++ node selected, the very top property in the right-hand pane is Additional Include Directories. That's exactly what you are looking for. (It's nice of them to put it on top.) Highlight the line for that choice and you'll notice the ellipsis button appears at the far right of the textbox. Click the ellipsis button to show Figure 10-11.

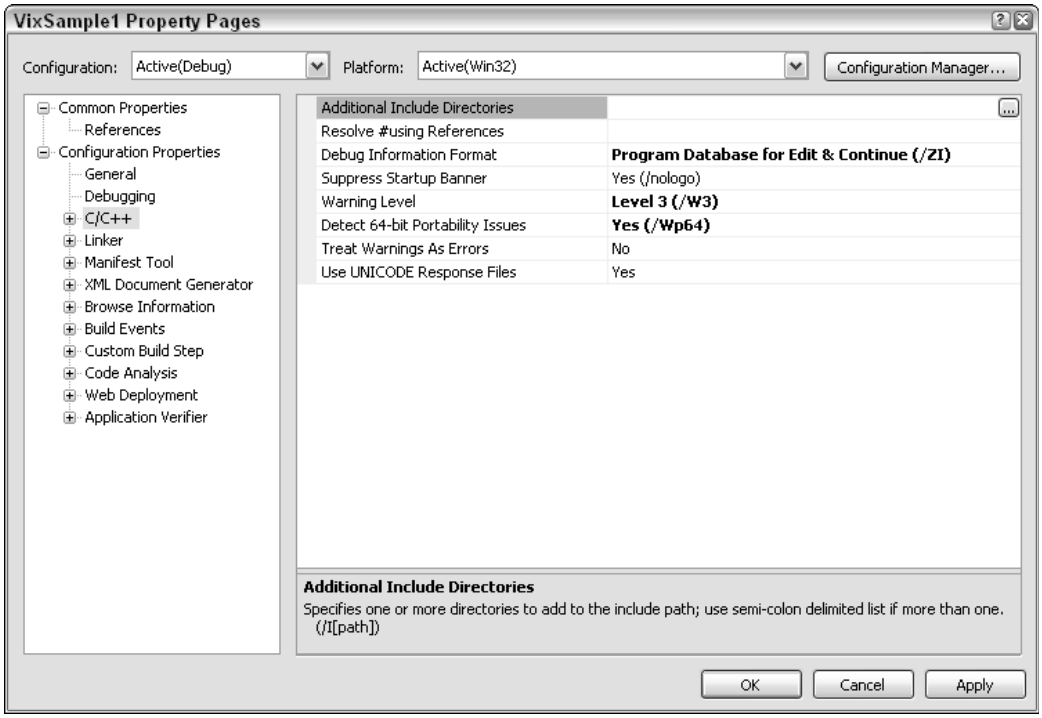

Figure 10-10

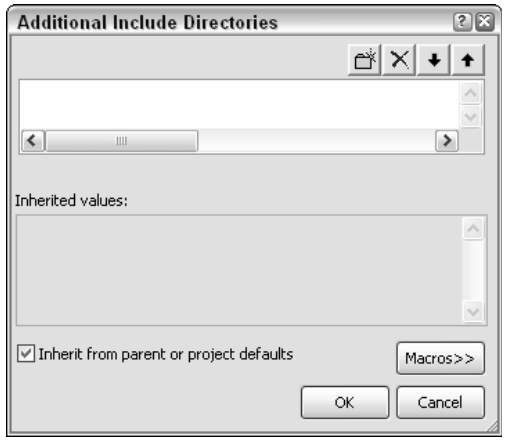

Figure 10-11

4. Now this is a very odd little window. It behaves very strangely for a window in Visual Studio 2005. As you can probably tell from the icons, it hasn't been updated in a while. Regardless, you need to add a new directory. Click the old-school new folder icon; it's the leftmost of the four icons located in the upper-right corner of the screen. This creates a new entry of sorts in the window directly below it.

5. It may seem strange, but there are two ways in which you can accomplish this next step:

   - You'll notice your cursor default into an edit mode inside the window. You can free hand type the path to the Vix headers now by entering `C:\Program Files\VMware\VMware VIX`.
   - For the second option, you can click yet another ellipsis button at the far right. If you click the ellipsis button, you get a standard open dialog box. Simply navigate to the location where you saved the Vix headers and click OK.

   Either method should result in Figure 10-12.

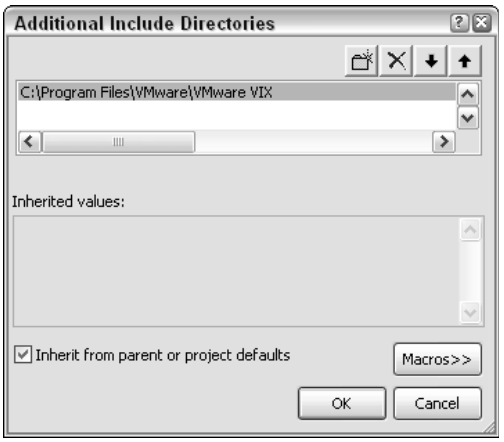

Figure 10-12

6. If your window looks like the preceding figure, click OK twice to get back out to the code editor.

The project now knows where to find the `vix.h` file you referenced in the `include` statement.

Pointing the Project to the Libraries Used

Finally, there's one more thing to do. You need to tell your project where the library file is to statically link the `vix.dll` file into your application:

1. Click the Project menu again and then the VixSample1 Properties item. This will open the project properties window once again. Expand the Configuration Properties node and then the Linker node. Finally, select Input to reveal Figure 10-13.

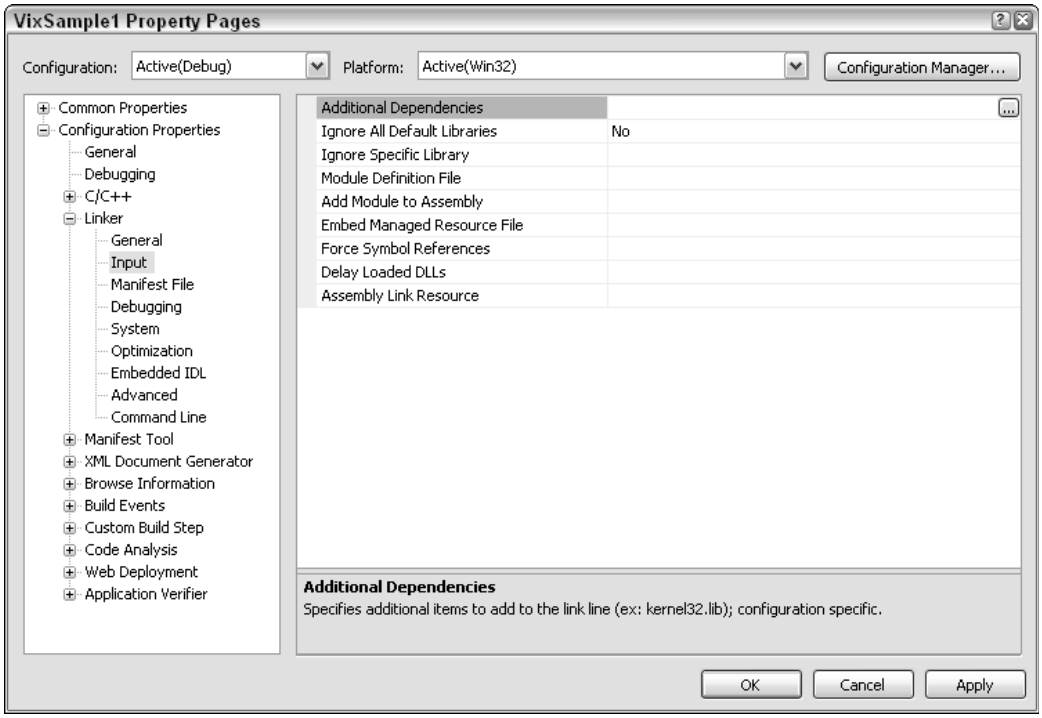

Figure 10-13

2. Again, the item you need is right at the top of the right-hand pane. (Nice of them to put it there once again!) Select the Additional Dependencies property and either click the ellipsis button or enter the following inside the property text area. If everything goes okay, you should see something similar to Figure 10-14.

```
"C:\Program Files\VMware\VMware Vix\vix.lib"
```

Be sure to enclose the path within double quotes. The property window will not do this for you. You will receive a linker error when you compile if you leave out the quotes.

3. Click OK to return to your project.

Every C application you write that leverages the Vix API will require this setup, so refer to these sections before you start that new bit of code.

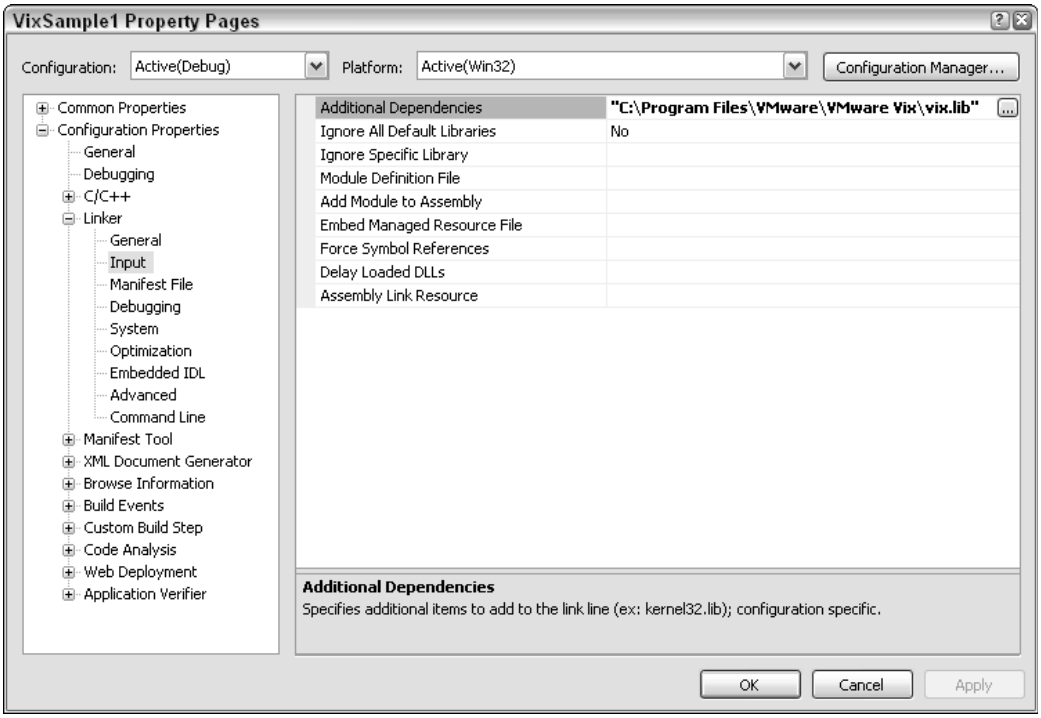

Figure 10-14

A Quick Vix Primer

Unlike the Perl and COM APIs, Vix is rather multilayered. There are several things that you must be aware of before you start poking and prodding your way through the first script. Though I'm not implying that poking and prodding is your normal method of coding, I nevertheless want to cover some basics to help you fully understand how Vix works.

Handles

Those of you familiar with C will undoubtedly have heard of a handle. A *handle* is little more than a pseudo-pointer, if you will. Without getting into when and why you might use a handle over a simple pointer, I'll just say the Vix API requires the use of handles of type `VixHandle` to reference almost all objects within the API.

The handles as they are used in Vix are more of what some might call a "magic cookie." This isn't some super secret programmer speak but a simple way of describing its use. The magic cookie gets passed around the code allowing you to uniquely identify and locate a specific object. In most cases this is done to mask operations performed against the handle, like any specific detail changes to the handle. This may be a difficult concept to grasp without specifics. The easiest generality to take away from this paragraph is that you need to use a handle to reference your objects in Vix. Take the following as a specific example:

```
VixHandle jobHandle = VIX_INVALID_HANDLE;

jobHandle = VixHost_Connect(VIX_API_VERSION,
                            VIX_SERVICEPROVIDER_VMWARE_SERVER,
                            NULL, // *hostName,
                            0, // hostPort,
                            NULL, // *userName,
                            NULL, // *password,
                            0, // options,
                            VIX_INVALID_HANDLE, // propertyListHandle,
                            NULL, // *callbackProc,
                            NULL); // *clientData);
```

The handle `jobHandle` in this case represents the object `VixHost_Connect`. Depending upon the type of object the handle represents, it will morph in a sense to certain properties and methods for that object. The actual content of the handle is nothing more than an integer. Internally Vix knows that the integer assigned to the handle refers to a specific object.

One last point on handles — every handle is reference counted. This means that when you created the `VixHost_Connect` object in the preceding code, the handle `jobHandle` received a reference count of 1. The importance of this is that because you have to manually release each and every handle you create, you must make sure the number of releases equals the reference count. A handle can be released by calling `Vix_ReleaseHandle(<handle name>)`. Take the following as an example:

```
VixHandle handle1;

handle1 = Vix_Host_Connect( stuff goes here )

// The handle now represents the Vix_Host_Connect object
// The handle has a reference count of 1

// Implement code to utilize the handle

//Release the handle
Vix_ReleaseHandle(handle1);
```

With that in mind if you assign more than one handle to the same object, only the first handle maintains the reference count because the second one merely refers to the first one. If that seems confusing, try this as an example:

```
VixHandle handle1;
VixHandle handle2;

handle1 = Vix_Host_Connect( parameters to connect )
//handle1 has a reference count of 1

Handle2 = Vix_Host_Connect( same parameters as above )  //same host is connected
//handle2 has the same value as handle1 (It represents the same object)
//handle1's reference count is now at 2 since there are two references to the
```

```
//object it represents.  Handle2 doesn't have a reference count since technically
//it's the same as handle1.

//Now, to release the handles you actually have to do it twice, once for each
//reference count

Vix_ReleaseHandle(handle1);

//At this point a valid handle still exists to the object so you can still use
//handle1.  Its reference count is now 1 again.

Vix_ReleaseHandle(handle1);

//Now the handle is gone.
```

Although this concept may seem a bit strange, it is an important concept to grasp. Any function that returns a handle can increase its reference count by one. Tracking this is important, as you will see in later examples.

Handle Types

There are four different types of handles that you will see throughout this chapter. Each handle type has a unique set of properties relating to the object they reference. Here's a quick list of the different types of handles you will encounter.

`VIX_HOST_HANDLE`	The host handle represents a host server running VMware Server. This kind of handle is commonly returned by the `VixHost_Open()` function.
`VIX_JOB_HANDLE`	The job handle comes about during any asynchronous operation in Vix. Because almost all the functions in the API are asynchronous in nature, you will find the job handle in almost everything you do.
`VIX_SNAPSHOT_HANDLE`	The snapshot handle is used to refer to a snapshot contained within a virtual machine. Currently the Vix API is very limited when it comes to snapshot handling.
`VIX_VM_HANDLE`	The virtual machine handle is used to refer to a specific virtual machine. The `VixVM_Open()` function provides you with this type of handle.

Handle Properties

As you can probably deduce, a handle can refer to many different types of objects within Vix. For instance, when a virtual machine object is referenced by a handle, that handle contains specific properties relating to that type of object. These properties are typed name/value pairs that can be any of the following types:

- 32-bit integer
- 64-bit integer
- String

- Boolean
- Handle

Handle properties are the way you modify and work with your object, regardless of their type. Their use ensures compatibility through any future version of the API. It's simply a layer of abstraction that you are most likely already familiar with.

The tables in the following short sections list for you the properties available for each type of handle. Unfortunately at this stage of release for Vix the descriptions of each property are not defined so I cannot provide you with more specific information on each one. You will see a few of these properties used throughout the rest of this chapter. Several in the following lists, however, have yet to be defined.

Virtual Machine Properties

These properties are available with handles of type `VIX_VM_HANDLE`.

Property
`VIX_PROPERTY_VM_NUM_VCPUS`
`VIX_PROPERTY_VM_VMX_PATHNAME`
`VIX_PROPERTY_VM_MEMORY_SIZE`
`VIX_PROPERTY_VM_POWER_STATE`
`VIX_PROPERTY_VM_TOOLS_STATE`
`VIX_PROPERTY_VM_IS_RUNNING`

Event Info Property

This property is present only during the `VixHost_FindItems()` function call.

Property
`VIX_PROPERTY_FOUND_ITEM_LOCATION`

Job Properties

These properties are available with handles of type `VIX_JOB_HANDLE`.

Property
`VIX_PROPERTY_JOB_RESULT_ERROR_CODE`
`VIX_PROPERTY_JOB_RESULT_VM_IN_GROUP`
`VIX_PROPERTY_JOB_RESULT_USER_MESSAGE`

Table continued on following page

Property
VIX_PROPERTY_JOB_RESULT_LINE_NUM
VIX_PROPERTY_JOB_RESULT_EXIT_CODE
VIX_PROPERTY_JOB_RESULT_COMMAND_OUTPUT
VIX_PROPERTY_JOB_RESULT_HANDLE

Snapshot Properties

These properties are available with handles of type `VIX_SNAPSHOT_HANDLE`.

Property
VIX_SNAPSHOT_DISPLAYNAME_PROPERTY
VIX_SNAPSHOT_DESCRIPTION_PROPERTY
VIX_SNAPSHOT_POWERSTATE_PROPERTY

The GetProperties() Function

Vix provides one function that allows you to retrieve properties from any handle. The `GetProperties()` function utilizes a `varargs` type signature that allows you to pass any number of arguments as long as you terminate the arguments with a special property, `VIX_PROPERTY_NONE`. Consider the following example:

```
VixError err;
VixHandle handle1;

int property1;
char* property2;

handle1 = <function that returns a handle>

err = Vix_GetProperties(handle1,
                        FIRST_PROPERTY_NAME,
                        &property1,
                        SECOND_PROPERTY_NAME,
                        &property2,
                        VIX_PROPERTY_NONE);

Vix_ReleaseHandle(handle1);
```

Sure that's a pretty generic example but the signature of the `Vix_GetProperties()` function is the important point here. You provide the handle followed by sets of name/value pairs and terminate it with the special `VIX_PROPERTY_NONE` property.

Error Codes

All Vix functions return an error code. The time in which it is returned is dependent upon the type of operation performed, synchronous or asynchronous. The error codes returned are of type `VixError` and are made up of a 64-bit value. If a function completes successfully, it returns an error code of `VIX_OK`, which is what you will find yourself checking for most often.

Now, the 64-bit error value that is returned comes with a bit of a catch. `VIX_OK` is the only error that you can check against the entire 64-bit value because it simply returns all zeros. All other errors only correspond to the least significant 16 bits of the error value, which are defined in a global type. In other words, the following code is valid:

```
VixError err;

err = VixJob_GetError(jobHandle);

if (VIX_OK == err) {
  // Success
}
```

This works because success is always all zeros. The least significant 16 bits do not need to be masked off from the rest of the code. That being said, the following is not valid:

```
VixError err;

err = VixJob_GetError(jobHandle);

if (VIX_E_INVALID_ARG == err) {
  // This doesn't work
}
```

Only success can be determined against the entire unmasked error code. So, you ask, how can you determine the error code if it's not success? Well, you can either mask the error code yourself or use one of three built-in functions to do it for you:

- First is `VIX_ERROR_CODE()`. This will mask the 64-bit value down to its least significant 16 bits for you. Using the preceding invalid bit of code let's make it valid:

```
VixError err;

err = VixJob_GetError(jobHandle);

if (VIX_E_INVALID_ARG == VIX_ERROR_CODE(err)) {
  // Works!  Handle the error code here
}
```

  Now instead of evaluating against the entire 64-bit error value you use `VIX_ERROR_CODE()` to chop off everything but the 16 least significant bits.

If you want to perform the masking yourself try the following:

```
VixError err;

err = VixJob_GetError(jobHandle);

if (VIX_E_INVALID_ARG == (err & 0xFFFF)) {
  // Works!  Performs the same masking that VIX_ERROR_CODE() does for you
}
```

Personally I prefer the built-in function; however, you can perform the masking manually if you wish.

- Next is the `VIX_SUCCEEDED()` function. This function performs the masking automatically and returns a Boolean of true if the error code is `VIX_OK`. Basically it says `(VIX_OK == err)` for you. Take the following example:

```
VixError err;

err = VixJob_GetError(jobHandle);

if (VIX_SUCCEEDED(err)) {
  // Handle successful return here
}
```

It's simple and perhaps a little cleaner and easier to read than the drawn out `VIX_OK == err`.

- Finally there's `VIX_FAILED()`. This function performs the bit masking and returns true if there is in fact an error. It's basically `(VIX_OK != err)` when you get down to it. The following example shows its usage:

```
VixError err;

err = VixJob_GetError(jobHandle);

if (VIX_FAILED(err)) {
  // Handle failure here
  // If you need to test against a specific failure this will not help you much.
}
```

One could argue their actual usefulness but I tend to use the built-in functions for evaluation when at all possible. That way if the error code structure is changed later I can be sure `VIX_SUCCESS()` *and* `VIX_FAILED()` *will always work. If you peek inside the* `vix.h` *header file you'll see that these are nothing more than built-in evaluations against* `VIX_OK`*. It's nothing more than a shortcut.*

There are numerous error values defined within the global type. I cover them in Appendix C.

Asynchronous Operations and the Job Handle

The topic of asynchronous versus synchronous operations is much discussed in the API chapters of this book. As you no doubt know, a *synchronous operation* executes and completes at that moment, whereas an *asynchronous operation* may take some time to complete. This topic fits into the handle discussion quite nicely.

As I've indicated, there are several different types of handles in Vix; however, the one most often used is referred to as a job handle. All asynchronous operations in Vix return a job handle. This handle represents the pending operation taking place and will, upon completion, be signaled with the final job status. Once they have been signaled to either the successful completion of a pending job or, worse, the failure of it, the job handle will possess several different properties that contain specifics about the completed job. The most important property is usually a handle that represents the final completed object — more on that in a minute.

Several approaches are available to determine the results of a pending job handle. I list them as follows and then talk about some brief examples:

- Polling for job completion (non-blocking)
- Blocking until the job is complete (blocking)
- Using callback functions (non-blocking)

Each has its place and depending upon your situation, you may end up using any combination of the three.

Polling for Job Completion

First I want to talk about polling for completion. When you register an asynchronous operation, as I have already explained, a job handle is created. Over time you will want to check, or poll, the resulting status of that operation. Vix provides a built-in function called `VixJob_CheckCompletion()`. This is a non-blocking function that sets a Boolean value representing the completion state of the asynchronous operation. You continue to poll the job handle with the function until you receive the completed value of true, at which time you can check the status and continue on with your code. Take the following as an example of polling an asynchronous operation.

> *All of the following examples are based upon a multithreaded environment. Vix is thread safe itself, and unless you are operating within a single-thread environment or require the use of single-threaded operations, the following will work. If you need more information about how to utilize Vix in a single-thread environment, please consult the VMware Programming API documentation.*

```
Bool openVMPoll(const VixHandle hostHandle,
            const char *vmName,
            VixHandle *vmHandle)
{
  VixError err;
  Vixhandle jobHandle = VIX_INVALID_HANDLE;

  if (!vmHandle) {
    return FALSE;
  }

  // Start the virtual machine

  jobHandle = VixVM_Open(hostHandle,
                         vmName,
                         NULL,
                         NULL);
```

```
    // Loop until complete is true meaning the job is complete
    for (Bool complete = FALSE; complete !=TRUE; ) {
      sleep(1);

      err = VixJob_CheckCompletion(jobHandle, &complete);

      if (VIX_FAILED(err)) {
        Vix_ReleaseHandle(jobHandle);
        return FALSE;
      }

      err = Vix_GetProperties(jobHandle,
                              VIX_PROPERTY_JOB_RESULT_HANDLE,
                              vmHandle,
                              VIX_PROPERTY_NONE);

      if (VIX_FAILED(err)) {
        Vix_ReleaseHandle(jobHandle);
        *vmHandle = VIX_INVALID_HANDLE;
        return FALSE;
      }

    }

    Vix_ReleaseHandle(jobHandle);
    return TRUE;

  }
```

This is a relatively simple example but rather powerful in terms of polling for job completion. First off the caller here must have already opened a connection to the host machine and must pass the corresponding handle in as `hostHandle`. Next the caller passes in the virtual machine's name, its configuration file path to be specific, and the handle he or she wants the resulting virtual machine object to reference once the job is complete.

Next the function makes sure the `vmHandle` provided is indeed valid and proceeds to try to connect to the virtual machine. This is an asynchronous operation, so the `VixVM_Open()` function is referenced by a `jobHandle`. The technique to poll for the job completion status is a simple loop. You loop as long as the complete Boolean value is false, which indicates the operation is not yet complete. You use the `VixJob_CheckCompletion()` function and sleep for one second between the end result, which you hope is that the `VixJob_CheckCompletion()` function finally returns a true and a successful result. Once this happens you use the `Vix_GetProperties()` function on the `jobHandle` to retrieve the virtual machine's handle via the property `VIX_PROPERTY_JOB_RESULT_HANDLE`. This sets the reference to the resulting virtual machine's handle to the `vmHandle` provided when the function was called.

That's your garden variety polling operation in Vix for a job handle. It will fit into most situations.

Blocking until the Job Is Complete

The next method utilizes a provided Vix function `VixJob_Wait()`. This function is a blocking function, meaning that all execution will halt on this command until it completes. All you have to provide to the

function is the job handle itself, and then once it returns, you check its status and retrieve the resulting handle from its properties. Take a look at this method in the following example. I'm going to morph the polling example I just used into one using the blocking function instead:

```
Bool openVMBlock(const VixHandle hostHandle,
                 const char *vmName,
                 VixHandle *vmHandle)
{

  VixError err;
  Vixhandle jobHandle = VIX_INVALID_HANDLE;

  if (!vmHandle) {
    return FALSE;
  }

  // Start the virtual machine

  jobHandle = VixVM_Open(hostHandle,
                         vmName,
                         NULL,
                         NULL);

  //Now, block with VixJob_Wait until complete
  err = VixJob_Wait(jobHandle,
                    VIX_PROPERTY_NONE);

  if (VIX_FAILED(err)) {
    Vix_ReleaseHandle(jobHandle);
    return FALSE;
  } else {
    err = Vix_GetProperties(jobHandle,
                            VIX_PROPERTY_JOB_RESULT_HANDLE,
                            vmHandle,
                            VIX_PROPERTY_NONE);

    if (VIX_FAILED(err)) {
      Vix_ReleaseHandle(jobHandle);
      *vmHandle = VIX_INVALID_HANDLE;
      return FALSE;
    }

  Vix_ReleaseHandle(jobHandle);
  return TRUE;

}
```

This example is set up exactly the same as before, except where you polled for the completion status before, now the function waits until it comes back, thus blocking the rest of the execution. Once it returns and is obviously successful, you retrieve the handle for the virtual machine, set the passed-in `vmHandle` to reference it, and exit the function. In reality it performs the same operation as before but in this case with the block while the asynchronous operation completes.

The `VixJob_Wait()` function also allows you to retrieve the resulting handle as a property without asking the job handle itself. An example of this would be if you modified the preceding code for the `VixJob_Wait()` function to this:

```
err = VixJob_Wait(jobHandle,
                  VIX_PROPERTY_JOB_RESULT_HANDLE,
                  &vmHandle,
                  VIX_PROPERTY_NONE);
```

Using a Callback Function

Last you have the callback. This is probably what you will find yourself using most often because it offers better performance for your code. All asynchronous functions in Vix offer a parameter to assign it a callback function. The prototype for any callback function is as follows:

```
typedef void (VixEventProc)(VixHandle handle,
                            VixEventType eventType,
                            VixHandle moreEventInfo,
                            void *clientData);
```

The thing to understand here is that the callback function, if provided, is registered with the job handle itself. Once the job handle is signaled by the asynchronous operation, an event is fired that calls the callback function. This doesn't always have to be at completion. There are four event types that cause a callback function to be initiated.

Event	Description
`VIX_EVENTTYPE_CALLBACK_SIGNALED`	Indicates that the operation has completed.
`VIX_EVENTTYPE_JOB_PROGRESS`	Reports progress on the operation. This may be fired several times during the operation.
`VIX_EVENTTYPE_FIND_ITEM`	Used by `VixHost_FindItems()`.
`VIX_EVENTTYPE_HOST_INITALIZED`	Used by `VixHost_Connect()`.

Most of the sample code provided in the "Functions" section that follows utilizes blocking functions to keep the examples simple. I do, however, use a callback to provide an example in the `VixHost_FindItems` function reference a little later in this chapter. If you explore this API to any great length, you will probably want to start leveraging the callback design for some tasks.

Functions

Now you get into the meat of things. These are general use functions that provide support while you are working with Vix. These will basically assist you with handle management, errors, and property retrieval.

> *Unless otherwise noted the examples provided are only to demonstrate usage and will not compile on their own.*

Vix_ReleaseHandle

This function reduces the reference count of a handle, or if only one reference count exists, it destroys the handle. The function prototype is provided in the following code block:

```
void Vix_ReleaseHandle(Vixhandle handle);
```

Parameters

Parameter	Description
`handle`	Any handle returned by any Vix function.

Example

```
VixHandle handle1 = VIX_INVALID_HANDLE;

//Implementation

Vix_ReleaseHandle(handle1);
```

The example here is rather simple, but illustrates the point of releasing your handles. Remember your reference counts and release your handles with `Vix_ReleaseHandle()` once you're finished. Every single script you write against this API will utilize this approach.

Vix_GetProperties

This function is used to retrieve the properties of the referenced handle. The properties available depend upon the type of object the handle references. This function also accepts parameters of type `varargs`. This means you can pass any number of parameters so long as you terminate the parameter list with the special `VIX_PROPERTY_NONE` property. The function prototype is provided in the following code block:

```
VixError Vix_GetProperties(VixHandle handle,
                           VixPropertyID propertyID,
                           ...);
```

Parameters

Parameter	Description
`handle`	Any handle returned by a Vix function.
`propertyID`	Any valid property ID for the type of object referenced.

Returns

The function returns a value of type `VixError`. Success is indicated by `VIX_OK`. Any failure can be accessed via the global error type covered later in Appendix C.

Example

```
VixError err;
VixHandle handle1 = VIX_INVALID_HANDLE;

//Function that assigns handle1 (ie. Connect to a host)

err = Vix_GetProperties(handle1,
                        propertyID,
                        &property1,
                        VIX_PROPERTY_NONE);

Vix_ReleaseHandle(handle1);
```

Though this example is rather abstract, I wanted to illustrate the use of `VIX_PROPERTY_NONE` to end the function's arguments. Most objects in Vix contain at least one type of property. I listed these earlier in this chapter. For instance, say you want to connect to a virtual machine and find out if it is in operation. The running status for the virtual machine is exposed via a property, `VIX_PROPERTY_VM_IS_RUNNING`. Given the handle for the virtual machine in question is `handle1`, you could write the following to retrieve that property:

```
bool isRunning;
VixError err;

err = Vix_GetProperties(handle1,
                        VIX_PROPERTY_VM_IS_RUNNING,
                        &isRunning,
                        VIX_PROPERTY_NONE);
```

You grab the status of the virtual machine, exposed as a `bool` by the `VIX_PROPERTY_VM_IS_RUNNING` property. The list of properties can continue on for as long as you have properties to give it. The trick is in the `VIX_PROPERTY_NONE` property. This simply signals the `Vix_GetProperties()` function that the list is finished; now give me my value.

Vix_GetPropertyType

This function identifies the type of property stored in a handle and referenced by the property ID provided. The following list outlines the possible values:

- 32-bit integer
- 64-bit integer
- String
- Boolean
- Handle

The function prototype is provided in the following code block:

```
VixError Vix_GetPropertyType(VixHandle handle,
                             VixPropertyID propertyID,
                             VixPropertyType *propertyType);
```

Parameters

Parameter	Description
`Handle`	Any handle returned by a Vix function.
`propertyID`	A valid property ID for the object referenced.
`propertyType`	The type of data for the property ID parameter.

Returns

The function returns a value of type `VixError`. Success is indicated by `VIX_OK`. Any failure can be accessed via the global error type covered later in Appendix C.

The `typedef` for `VixPropertyType` can be found in the `vix.h` header file. Basically all this function does for you is allow you to determine the data type returned via a given property. A property that is a 32-bit integer will return a `VIX_PROPERTYTYPE_INTEGER`, or an integer value of 1. The following is the `typedef` for the `VixPropertyType`:

```
typedef int VixPropertyType;
enum {
  VIX_PROPERTYTYPE_ANY       = 0,
  VIX_PROPERTYTYPE_INTEGER   = 1,
  VIX_PROPERTYTYPE_STRING    = 2,
  VIX_PROPERTYTYPE_BOOL      = 3,
  VIX_PROPERTYTYPE_HANDLE    = 4,
  VIX_PROPERTYTYPE_INT64     = 5,
}
```

Vix_GetHandleType

This function returns the type of handle provided as the parameter. The list that follows outlines the six different types of handles available:

- `VIX_HANDLETYPE_NONE`
- `VIX_HANDLETYPE_HOST`
- `VIX_HANDLETYPE_VM`
- `VIX_HANDLETYPE_NETWORK`
- `VIX_HANDLETYPE_JOB`
- `VIX_HANDLETYPE_SNAPSHOT`

The function prototype is provided in the following code block:

```
VixHandleType Vix_GetHandleType(VixHandle handle);
```

Parameters

Parameter	Description
`handle`	Any handle returned by a Vix function.

Returns

An enumerated type from the preceding list indicating what type of handle was passed in.

Example

```
VixHandle handle1 = VIX_INVALID_HANDLE;

//Function that assigns the handle1 (ie. Connect to a host)

if (Vix_GetHandleType(handle1) == VIX_HANDLETYPE_HOST) {
  //implement logic here
}

Vix_ReleaseHandle(handle1)
```

This function allows you to determine the type of handle you have, if for some reason you don't know. The return is based upon an enumerated type `VixHandleType` shown in the following code block:

```
typedef int VixHandleType;
enum {
  VIX_HANDLETYPE_NONE      = 0,
  VIX_HANDLETYPE_HOST      = 2,
  VIX_HANDLETYPE_VM        = 3,
  VIX_HANDLETYPE_NETWORK   = 5,
  VIX_HANDLETYPE_JOB       = 6,
  VIX_HANDLETYPE_SNAPSHOT  = 7,
}
```

Vix_GetErrorText

This function allows you to retrieve the string description of the error passed in. The function prototype is provided in the following code block:

```
const char * Vix_GetErrorText(VixError err,
                              const char *locale);
```

Parameters

Parameter	Description
`err`	The actual error as type `VixError`.
`locale`	Currently not implemented. You must pass a NULL to this parameter.

Returns

This function returns a string that describes the error held by the passed in `VixError` object.

Example

```
VixHandle handle1 = VIX_INVALID_HANDLE;
VixError err;
const char* errText

//Function that assigns the handle1 (ie. Connect to a host)
//Blocking function (ie. VixJob_Wait) that returns a VixError object.

errText = Vix_GetErrorText(err,
                           NULL);
```

This is again a little abstract, but the idea is alive. You want to perform some function, say connecting to a VMware Server host. In your blocking function, which you will see later in this chapter, you assign its result to `err`, which is of type `VixError`. The `Vix_GetErrorText()` function takes this `VixError` object and returns to you a string containing the human-readable text of that error. The most common use for this function is error logging. Checking for specific errors within your code can be accomplished using the error codes listed in Appendix C; there's little use for the actual error message within your code. If you were to log output from your script, however, this function gains some use in that you can dump out the actual error text in a human-readable way.

Vix_FreeBuffer

This function is provided to allow you to free any string buffer that certain Vix functions create automatically. For example, if you retrieve the string that contains the location of the virtual machine's configuration file, the `Vix_GetProperties` function allocates the space for that string when it is returned. It is your responsibility to free that string buffer once you are finished. The function prototype is provided in the following code block:

```
void Vix_FreeBuffer(void *p);
```

Parameters

Parameter	Description
`p`	Any string pointer returned by a Vix function.

Example

```
VixHandle handle1 = VIX_INVALID_HANDLE;
VixError err;
char *configLocation;

//Open a virtual machine and retrieve a handle.

err = Vix_GetProperties(handle1,
                        VIX_VM_VMX_PATHNAME,
```

```
                          &configLocation,
                          VIX_PROPERTY_NONE);

  printf("The location is:%p\n", configLocation);

  Vix_FreeBuffer(configLocation)
```

Here you open a virtual machine already present on your VMware Server host and query the `VIX_VM_VMX_PATHNAME` property to determine its location. This location, represented by `configLocation`, is allocated by the `Vix_GetProperties()` function and must be freed after you are finished. A little housekeeping is all.

Vix_CreateTemporaryObject

This function allows you to create a temporary object. That may seem vague, but in version 1.0 of the Vix API, there is only one type of temporary object that exists, the property list. The function prototype is provided in the following code block:

```
VixError Vix_CreateTemporaryObject(VixHandleType objectType,
                                   VixHandle *resultObject);
```

Parameters

Parameter	Description
`objectType`	For this release the `objectType` must be of type `VIX_PROPERTY_LIST_HANDLE`.
`resultObject`	A pointer to the handle that will reference the resultant temporary object.

The use of this function is still a little vague. Perhaps the next iteration of Vix will find more use for this function.

Vix_PumpEvents

This function is used during single-thread operations. As I stated before, this chapter does not deal with operations in a single-thread environment; however, I provide the information here as a reference. For more information visit the Vix API documentation. The function prototype is provided in the following code block:

```
void Vix_PumpEvents(VixHandle hostHandle,
                    VixPumpEventsOptions options);
```

Parameters

Parameter	Description
`handle`	This references the handle to the host object.
`options`	In this release the `options` parameter must be 0.

I use the `Vix_PumpEvents()` function later in this chapter when I cover callbacks. So to keep things simple for now I'll hold off on the example, and you can see it in practice later.

Host Machine Operations

The host machine, like the other APIs, refers to the VMware Server machine itself. Actions such as connecting to, disconnecting from, registering, and deregistering virtual machines are a few of the functions you will find in this section.

> *Unless otherwise noted the examples provided are only to demonstrate usage and will not compile on their own.*

VixHost_Connect

This function is used to connect to a local or remote VMware Server. This is by far the most used function because all operations require connection to a host machine. The function prototype is provided in the following code block:

```
VixHandle VixHost_Connect(int apiVersion,
                          VixServiceProvider hostType,
                          const char *hostName,
                          int hostPort,
                          const char *userName,
                          const char *password,
                          VixHostOptions options,
                          VixHandle propertyListHandle,
                          VixEventProc *callbackProc,
                          void *clientData);
```

Parameters

Parameter	Description
`apiVersion`	The API version to be supported. There is only one to use at this time, `VIX_API_VERSION`.
`hostType`	The type of host being opened. This version of the API supports only VMware Server, so you must pass `VIX_SERVICEPROVIDER_VMWARE_SERVER`.

Table continued on following page

Parameter	Description
hostName	Name of the host you want to connect to. If you pass NULL the local host is assumed.
hostPort	Port number that VMware Server is set up to communicate on. Typically port 902; however, 0 is assumed to be the local host.
username	The user name you wish to authenticate with. A value of NULL will authenticate using the current user.
Password	The password that goes with the user name provided by the preceding parameter. Assign NULL if you are using the current user to authenticate.
Options	Only one option exists, which is VIX_HOSTOPTION_USE_EVENT_PUMP. You would use this in a single-threaded environment. Otherwise pass NULL.
propertyListHandle	At this time this must be VIX_INVALID_HANDLE.
callbackProc	The name of the callback function to be invoked when the operation completes.
clientData	This is a parameter that will be passed to the callbackProc listed previously.

Returns

This function returns a job handle of type VixHandle. Once the operation is complete, you can retrieve the host object handle from the job handle returned.

Example

```
VixError err;
VixHandle hostHandle = VIX_INVALID_HANDLE;
VixHandle jobHandle = VIX_INVALID_HANDLE;

jobHandle = VixHost_Connect(VIX_API_VERSION,
                            VIX_SERVICEPROVIDER_VMWARE_SERVER,
                            NULL, //hostname
                            0, //port
                            NULL, //username
                            NULL, //password
                            0, //options
                            VIX_INVALID_HANDLE, //propertyListHandle
                            NULL, //callbackProc
                            NULL); //clientData

err = VixJob_Wait(jobHandle,
                  VIX_PROPERTY_JOB_RESULT_HANDLE,
                  &hostHandle,
                  VIX_PROPERTY_NONE);
```

```
//Do stuff

Vix_ReleaseHandle(jobHandle);
Vix_ReleaseHandle(hostHandle)
```

This example simply demonstrates how you would use `VixHost_Connect()` to connect to a VMware Server located on the local host. You can replace the `hostname`, `port`, `username`, and `password` parameters to connect to a remote VMware Server. It's one of the simpler things to do.

VixHost_Disconnect

This function is used to disconnect from a host that was connected to with the preceding function. One important exception to note with this function — using this function on your host handle takes care of releasing the handle for you. There is no need to call `Vix_ReleaseHandle()` after a successful `VixHost_Disconnect()`. The function prototype is provided in the following code block:

```
void VixHost_Disconnect(VixHandle hostHandle);
```

Parameters

Parameter	Description
`handle`	This is the host handle that references the host you wish to disconnect from. This handle was retrieved from the job handle after the asynchronous operation was complete.

Example

```
VixHandle hostHandle = VIX_INVLAID_HANDLE;
VixHandle jobHandle = VIX_INVALID_HANDLE;
VixError err;

jobHandle = VixHost_Connect( <parameters to connect> )

err = VixJob_Wait(jobHandle,
                  VIX_PROPERTY_JOB_RESULT_HANDLE,
                  &hostHandle,
                  VIX_PROPERTY_NONE);

if (VIX_FAILED(err)) {
  //Handle error
}

Vix_ReleaseHandle(jobHandle)

// Do stuff with the host

VixHost_Disconnect(hostHandle);

//Do not call Vix_ReleaseHandle on the hostHandle at this point.  It is not
//necessary.
```

All objects eventually cease to exist. The `VixHost_Disconnect()` function does this for you. When given a handle that refers to a host, it will not only disconnect from that host, but it will also release the handle. Let me say that again, there's no reason to release the handle if you call `VixHost_Disconnect()`. It is one of the only functions in Vix that handles the handle release for you.

VixHost_FindItems

This function's sole purpose in this release is to find running virtual machines on a given host and call the defined callback function. Note that I said running and not just registered. If no virtual machines are currently powered on, this function will return nothing. I provide a working example of its use with a callback in the example that follows. Remember to use the development environment I outlined at the beginning of the chapter if you compile and test this example. The function prototype is provided in the following code block:

```
VixHandle VixHost_FindItems(VixHandle hostHandle,
                            VixFindItemType searchType,
                            VixHandle searchCriteria,
                            int32 timeout,
                            VixEventProc *callbackProc,
                            void *clientData);
```

`The callback function has the following prototype:`

```
typedef void (VixEventProc)(VixHandle handle,
                            VixEventType eventType,
                            VixHandle moreEventInfo,
                            void *clientData);
```

Parameters

Parameter	Description
`hostHandle`	The host handle that represents the host machine you are performing the search against.
`searchType`	Must be `VIX_FIND_RUNNING_VMS` for this release of the API.
`searchCriteria`	Must be `VIX_INVALID_HANDLE` in this release of the API.
`timeout`	Must be -1 for this release of the API.
`callbackProc`	The name of the function to invoke when `VixHost_FindItems` fires its events.
`clientData`	User-defined data to be passed to the callback function.

Returns

This function is asynchronous and returns a job handle.

Example

```
#include "stdafx.h"
#include "vix.h"

static VixHandle hostHandle = VIX_INVALID_HANDLE;

// This is the callback function
void VixFindVMS(VixHandle jobHandle,
                VixEventType eventType,
                VixHandle eventInfo,
                void *clientData)
{
  VixError err;
  char *path = NULL;

  //Make sure we only do something if the callback event is FIND_ITEM
  if (VIX_EVENTTYPE_FIND_ITEM != eventType) {
    return;
  }

  //Something was found if we are this far.  Do something with it.
  err = Vix_GetProperties(eventInfo,
                          VIX_PROPERTY_FOUND_ITEM_LOCATION,
                          &path,
                          VIX_PROPERTY_NONE);

  if (VIX_FAILED(err)) {
    //Handle an error
    goto abort;
  }

  printf("Found virtual machine: %s\n", path);

abort:
  Vix_FreeBuffer(path);
}

//Main function
int _tmain(int argc, _TCHAR* argv[])
{
  VixHandle jobHandle = VIX_INVALID_HANDLE;
  VixHandle hostHandle = VIX_INVALID_HANDLE;
  VixError err;

  //This connects to a VMware Server instance on the local host.
  //If your server is remote change the hostname, port, username, and password
  //items in the parameter list.
  jobHandle = VixHost_Connect(VIX_API_VERSION,
                              VIX_SERVICEPROVIDER_VMWARE_SERVER,
                              NULL, //hostname
                              0, //port
                              NULL, //username
                              NULL, //password
                              0,
                              VIX_INVALID_HANDLE,
```

```
                                NULL,
                                NULL);

    err = VixJob_Wait(jobHandle,
                      VIX_PROPERTY_JOB_RESULT_HANDLE,
                      &hostHandle,
                      VIX_PROPERTY_NONE);

    if (VIX_FAILED(err)) {
      printf("\nERROR\n");
      goto abort;
    }

    Vix_ReleaseHandle(jobHandle);

    printf("Here is a list of running virtual machines.\n");

    jobHandle = VixHost_FindItems(hostHandle,
                                  VIX_FIND_RUNNING_VMS,
                                  VIX_INVALID_HANDLE,
                                  -1,
                                  VixFindVMS,
                                  NULL);

    VixJob_Wait(jobHandle, VIX_PROPERTY_NONE);

  abort:
    Vix_ReleaseHandle(jobHandle);
    VixHost_Disconnect(hostHandle);

  }
```

Here I make good use of a callback function along with the `VixHost_FindItems()` function to retrieve a list of active virtual machines. This release of Vix makes things a little confusing for this function. If you remember from the prototype for `VixHost_FindItems`, the search type can only be one thing, `VIX_FIND_RUNNING_VMS`, the `searchCriteria` must be `VIX_INVALID_HANDLE` because it has yet to be implemented, the `timeout` must be `-1`, and the `clientData` in this case is `NULL` because you don't need it. With those restrictions there's little left to do with this function beyond the preceding example.

First you start off in the `Main()` function. Here you connect to a VMware Server host and retrieve the `hostHandle` from the blocking function `VixJob_Wait()`. Once you call the host handle, you pass it along with the mandatory values outlined in the preceding paragraph and the callback function name to `VixHost_FindItems()`. As you can see, the callback function is named `VixFindVMS()`. Moving back up the code block, you'll find the `VixFindVMS()` function. Now, an important concept to grasp is when this callback is actually called. The answer is, quite a bit. Several different events are fired during the execution of a single function. The trick is to capture and process only the events you are interested in. In this case you only care if the callback is triggered with a `VIX_EVENTTYPE_FIND_ITEM`. When the `VixHost_FindItems()` function triggers the callback, it passes in the event type that triggered it as a `VixEventType` object. You evaluate this to determine if you need to process this event (`VIX_EVENT TYPE_FIND_ITEM != eventType`). If this is true, then you don't want this event, and you kick out of the function. If false, it is the right event, and you continue on. Next you grab the properties from the

eventType object, retrieve the virtual machine's location, and print it out. It's all about events and understanding when and how they are triggered.

VixHost_RegisterVM

This function, as you can tell, adds a virtual machine to the host server's inventory. The function prototype is provided in the following code block:

```
VixHandle VixHost_RegisterVM(VixHandle hostHandle,
                             const char *vmxFilePath,
                             VixEventProc *callbackProc,
                             void *clientData);
```

Parameters

Parameter	Description
hostHandle	The host handle that represents the host machine you are working with.
vmxFilePath	The path of the virtual machine's configuration file you wish to add. The format of the path depends upon the type of host operating system in the VMware Server.
callbackProc	A function to be called for VixHost_RegisterVM events.
clientData	User-supplied data that is passed to the callback function.

Returns

This function is asynchronous and returns a job handle.

Example

```
VixHandle hostHandle = VIX_INVALID_HANDLE;
VixHandle jobHandle = VIX_INVALID_HANDLE;
VixError err;

// Connect to a VMware Server and retrieve a host handle

jobHandle = VixHost_RegisterVM(hostHandle,
                               "C:\\VirtualMachines\\test-winxppro-sp2\\
                                test-winxppro-sp2.vmx",
                               NULL,
                               NULL);

err = VixJob_Wait(jobHandle,
                  VIX_PROPERTY_NONE);

//Evaluate success
```

Again this is abstract but provides an illustration of the function's use. The keys are having a valid host handle for your VMware Server and the path to the configuration file you wish to register. If the virtual machine is already registered, it will just return a VM_OK and call it a success.

VixHost_UnregisterVM

This function obviously allows you to deregister a virtual machine from the host server's inventory. The function prototype is provided in the following code block:

```
VixHandle VixHost_UnregisterVM(VixHandle hostHandle,
                               const char *vmxFilePath,
                               VixEventProc *callbackProc,
                               void *clientData);
```

Parameters

Parameter	Description
`hostHandle`	The host handle that represents the host machine you are working with.
`vmxFilePath`	The path of the virtual machine's configuration file you wish to remove. The format of the path depends upon the type of host operating system in the VMware Server.
`callbackProc`	A function to be called for `VixHost_UnregisterVM` events.
`clientData`	User-supplied data that is passed to the callback function.

Returns

This function is asynchronous and returns a job handle.

Example

```
VixHandle hostHandle = VIX_INVALID_HANDLE;
VixHandle jobHandle = VIX_INVALID_HANDLE;
VixError err;

// Connect to a VMware Server and retrieve a host handle

jobHandle = VixHost_UnregisterVM(hostHandle,
                                 "C:\\VirtualMachines\\test-winxppro-sp2\\↩
                                  test-winxppro-sp2.vmx",
                                 NULL,
                                 NULL);

err = VixJob_Wait(jobHandle,
                  VIX_PROPERTY_NONE);

//Evaluate success
```

This is exactly the opposite from the `VixHost_RegisterVM()` code example. The same procedure applies with the exception that you are removing a virtual machine from inventory. This operation will succeed only if the virtual machine is not powered on. If the virtual machine is not in the inventory, it will still succeed.

Virtual Machine Operations

This section covers the different types of functions available to work with your virtual machines. All the functions here require a valid host handle, in other words a connection to a server, in order to work.

> *Unless otherwise noted the examples provided are only to demonstrate usage and will not compile on their own.*

VixVM_Open

This is where it all begins. Once you have a valid host handle one of the first things you have to do before working with a virtual machine is to open it. This function takes the host handle and a path to the virtual machine and gives back a virtual machine handle. This virtual machine handle is required for most other virtual machine operations. The function prototype is provided in the following code block:

```
VixHandle VixVM_Open(VixHandle hostHandle,
                     const char *vmxFilePathName,
                     VixEventProc *callbackProc,
                     void *clientData);
```

Parameters

Parameter	Description
`hostHandle`	The handle that represents the host object you are currently using.
`vmxFilePathName`	The path to the virtual machine configuration file.
`callbackProc`	The callback function that will be called for the function's events.
`clientData`	User data to be passed to the callback function.

Returns

This is an asynchronous call and returns a job handle.

Example

```
VixError err;
VixHandle jobHandle = VIX_INVALID_HANDLE;
VixHandle hostHandle = VIX_INVALID_HANDLE;
VixHandle vmHandle = VIX_INVALID_HANDLE;

// Use VixHost_Connect and the jobHandle to retrieve a valid hostHandle

jobHandle = VixVM_Open(hostHandle,
                       "C:\\virtualmachines\\dev-winxppro-sp2\\↩
                        dev-winxppro-sp2.vmx",
                       NULL,
                       NULL);
```

```
err = VixJob_Wait(jobHandle,
                  VIX_PROPERTY_JOB_RESULT_HANDLE,
                  &vmHandle,
                  VIX_PROPERTY_NONE);

//Release handles
```

This function is how you get a new virtual machine handle. How often you use this depends a lot on what you do with the API, but chances are you'll be in this function from time to time. You simply need a handle to your host and the path to the virtual machine you want to open. This will return a virtual machine handle that you can then use to power on and perform other tasks against.

VixVM_PowerOn

This one needs little explanation to its purpose. The function powers on a virtual machine that is currently off or resumes a virtual machine that is currently suspended. The function prototype is provided in the following code block:

```
VixHandle VixVM_PowerOn(vixHandle vmHandle,
                        VixVMPowerOpOptions powerOnOptions,
                        VixHandle propertyListHandle,
                        VixEventProc *callbackProc,
                        void *clientData);
```

Parameters

Parameter	Description
`vmHandle`	The handle that represents the virtual machine you want to power on. You create this handle by calling the `VixVM_Open()` function.
`powerOnOptions`	For this release the only option supported is `VIX_VMPOWEROP_NORMAL`.
`propertyListHandle`	Must be `VIX_INVALID_HANDLE`.
`calbackProc`	A callback function that's called for the function's events.
`clientData`	User-provided data that gets passed to the callback function.

Returns

This is an asynchronous call and returns a job handle.

Example

```
VixError err;
VixHandle jobHandle = VIX_INVALID_HANDLE;
VixHandle hostHandle = VIX_INVALID_HANDLE;
VixHandle vmHandle = VIX_INVALID_HANDLE;

// Use VixHost_Connect and the jobHandle to retrieve a valid hostHandle

// Use VixVM_Open to connect to a virtual machine and retrieve a vmHandle
```

```
jobHandle = VixVM_PowerOn(vmHandle,
                          VIX_VMPOWEROP_NORMAL,
                          VIX_INVALID_HANDLE,
                          NULL,
                          NULL);

err = VixJob_Wait(jobHandle, VIX_PROPERTY_NONE);

//Release handles
```

Here you see where you need that virtual machine handle from the previous section. The only possible power operation mode for this release of Vix is `VIX_VMPOWEROP_NORMAL`, so provide your virtual machine handle and powerop mode and you're off.

VixVM_PowerOff

This function powers off a currently running virtual machine. If the machine is not running or currently suspended it will return a `VIX_E_VM_NOT_RUNNING` error. The function prototype is provided in the following code block:

```
VixHandle VixVM_PowerOff(VixHandle vmHandle,
                         VixVMPowerOpOptions powerOffOptions,
                         VixEventProc *callbackProc,
                         void *clientData);
```

Parameters

Parameter	Description
`vmHandle`	The handle that represents the virtual machine you want to power off. You create this handle by calling the `VixVM_Open()` function.
`powerOffOptions`	Must be zero (0) in this release.
`callbackProc`	A callback function that's called for the function's events.
`clientData`	User-provided data that gets passed to the callback function.

Returns

This is an asynchronous call and returns a job handle.

Example

```
VixError err;
VixHandle jobHandle = VIX_INVALID_HANDLE;
VixHandle hostHandle = VIX_INVALID_HANDLE;
VixHandle vmHandle = VIX_INVALID_HANDLE;

// Use VixHost_Connect and the jobHandle to retrieve a valid hostHandle

// Use VixVM_Open to connect to a virtual machine and retrieve a vmHandle
```

```
// Assuming the virtual machine is currently running.
jobHandle = VixVM_PowerOff(vmHandle,
                           0,
                           NULL,
                           NULL);

err = VixJob_Wait(jobHandle, VIX_PROPERTY_NONE);

\\Evaluate error here
if (Vix_Failed(err)) {
  \\Handle error and check to see if the error was VIX_E_VM_NOT_RUNNING
}

\\Release Handles
```

Unless you are utilizing a callback, there's only one way to do this. Call the `VixVM_PowerOff()` function, provide your virtual machine handle and power off option, which must be zero (0), and it powers off the machine. Make sure you carefully check the error status returned from the job handle because you can't power off a machine that's not running. Such an attempt will result in an error.

VixVM_Reset

This function performs a reset of the virtual machine. If the virtual machine is currently powered on, it resets and reboots the machine. If the machine is off an error occurs. The function prototype is provided in the following code block:

```
VixHandle VixVM_Reset(VixHandle vmHandle,
                      VixVMPowerOpOptions powerOnOptions,
                      callbackProc *callbackProc,
                      void *clientData);
```

Parameters

Parameter	Description
`vmHandle`	The handle that represents the virtual machine you want to reset. You create this handle by calling the `VixVM_Open()` function.
`powerOnOptions`	This release requires this to be `VIX_VMPOWEROP_NORMAL`.
`callbackProc`	A callback function that's called for the function's events.
`clientData`	User-provided data that gets passed to the callback function.

Returns

This is an asynchronous call and returns a job handle.

Example

```
VixError err;
VixHandle jobHandle = VIX_INVALID_HANDLE;
VixHandle hostHandle = VIX_INVALID_HANDLE;
VixHandle vmHandle = VIX_INVALID_HANDLE;

// Use VixHost_Connect and the jobHandle to retrieve a valid hostHandle

// Use VixVM_Open to connect to a virtual machine and retrieve a vmHandle

// Assuming the virtual machine is currently running.
jobHandle = VixVM_Reset(vmHandle,
                        VIX_VMPOWEROP_NORMAL,
                        NULL,
                        NULL);

err = VixJob_Wait(jobHandle, VIX_PROPERTY_NONE);

\\Evaluate error here
if (Vix_Failed(err)) {
  \\Handle error and check to see if the error was VIX_E_VM_NOT_RUNNING
}
```

This function is very similar to the `VixVM_PowerOn()` function. It takes a virtual machine handle and a power operation mode. Liken this to pressing the reset button on a physical computer. It's a warm boot if you will.

VixVM_Suspend

This function suspends the virtual machine referenced. If the virtual machine is not powered on an error occurs. The function prototype is provided in the following code block:

```
VixHandle VixVM_Suspend(VixHandle vmHandle,
                        VixVMPowerOpOptions powerOffOptions,
                        VixEventProc *callbackProc,
                        void *clientData);
```

Parameters

Parameter	Description
`vmHandle`	The handle that represents the virtual machine you want to suspend. You create this handle by calling the `VixVM_Open()` function.
`powerOffOptions`	Must be zero (0) in this release.
`callbackProc`	A callback function that's called for the function's events.
`clientData`	User-provided data that gets passed to the callback function.

Returns

This is an asynchronous call and returns a job handle.

Example

```
VixError err;
VixHandle jobHandle = VIX_INVALID_HANDLE;
VixHandle hostHandle = VIX_INVALID_HANDLE;
VixHandle vmHandle = VIX_INVALID_HANDLE;

// Use VixHost_Connect and the jobHandle to retrieve a valid hostHandle

// Use VixVM_Open to connect to a virtual machine and retrieve a vmHandle

// Assuming the virtual machine is currently running.
jobHandle = VixVM_Suspend(vmHandle,
                          0,
                          NULL,
                          NULL);

err = VixJob_Wait(jobHandle, VIX_PROPERTY_NONE);

\\Evaluate error here
if (Vix_Failed(err)) {
  \\Handle error and check to see if the error was VIX_E_VM_NOT_RUNNING
}
```

You may have noticed that the past several examples have a striking similarity. I point this out only to illustrate how easy this API is to use. Like before you provide a virtual machine handle and this time the power off options, which in this release can only be zero (0). If the virtual machine is powered on and not already suspended, it will suspend it to disk. If the machine is off, the function will throw an error, so be sure and check your error status.

VixVM_Delete

This function deletes a virtual machine; however, it doesn't just remove it from inventory. It deletes all the files on the host system. *Use this function with extreme caution.* If the virtual machine is not powered off or in a suspended state an error occurs. The function prototype is provided in the following code block:

```
VixHandle VixVM_Delete(VixHandle vmHandle,
                       VixVMDeleteOptions deleteOptions,
                       VixEventProc *callbackProc,
                       void *clientData);
```

Parameters

Parameter	Description
`vmHandle`	The handle that represents the virtual machine you want to delete. You create this handle by calling the `VixVM_Open()` function.
`deleteOptions`	Must be zero (0) in this release.

Parameter	Description
`callbackProc`	A callback function that's called for the function's events.
`clientData`	User-provided data that gets passed to the callback function.

Returns

This is an asynchronous call and returns a job handle.

Example

```
VixError err;
VixHandle jobHandle = VIX_INVALID_HANDLE;
VixHandle hostHandle = VIX_INVALID_HANDLE;
VixHandle vmHandle = VIX_INVALID_HANDLE;

// Use VixHost_Connect and the jobHandle to retrieve a valid hostHandle

// Use VixVM_Open to connect to a virtual machine and retrieve a vmHandle

// Assuming the virtual machine is currently running.
jobHandle = VixVM_Delete(vmHandle,
                         0,
                         NULL,
                         NULL);

err = VixJob_Wait(jobHandle, VIX_PROPERTY_NONE);

\\Evaluate error here
if (Vix_Failed(err)) {
  \\Handle error and check to see if the error was VIX_E_VM_NOT_RUNNING
}
```

I need to say it again — *use this function with extreme caution.* Remember, it doesn't just remove the virtual machine from inventory; it removes it from disk as well. Your virtual machine is gone, completely, after this asynchronous function completes. As with the past several examples, provide a virtual machine handle and in this case the delete options, which can only be zero (0) in this release.

VixVM_CreateSnapshot

This function creates a new snapshot for the referenced virtual machine. This release of the API supports only a single snapshot per virtual machine. If you attempt to create more than one through the API, the previous snapshot will be overwritten. You can create a snapshot in either a power on or off state. The function prototype is provided in the following code block:

```
VixHandle VixVM_CreateSnapshot(VixHandle vmHandle,
                               const char *name,
                               const char *description,
                               int options,
                               VixHandle propertyListHandle,
                               VixEventProc *callbackProc,
                               void *clientData);
```

Parameters

Parameter	Description
`vmHandle`	The handle that represents the virtual machine you want to create a snapshot for. You create this handle by calling the `VixVM_Open()` function.
`name`	Must be NULL in this release.
`description`	Must be NULL in this release.
`options`	Must be zero (0) in this release.
`propertyListHandle`	Must be `VIX_INVALID_HANDLE`.
`callbackProc`	A callback function that's called for the function's events.
`clientData`	User-defined data that gets passed to the callback function.

Returns

This is an asynchronous call and returns a job handle.

Example

```
VixError err;
VixHandle jobHandle = VIX_INVALID_HANDLE;
VixHandle hostHandle = VIX_INVALID_HANDLE;
VixHandle vmHandle = VIX_INVALID_HANDLE;
VixHandle snapshotHandle = VIX_INVALID_HANDLE;

// Use VixHost_Connect and the jobHandle to retrieve a valid hostHandle

// Use VixVM_Open to connect to a virtual machine and retrieve a vmHandle

// The virtual machine can be powered on or off at this point.

jobHandle = VixVM_CreateSnapshot(vmHandle,
                                 NULL,
                                 NULL,
                                 0,
                                 VIX_INVALID_HANDLE,
                                 NULL,
                                 NULL);

err = VixJob_Wait(jobHandle,
                  VIX_PROPERTY_JOB_RESULT_HANDLE,
                  &snapshotHandle,
                  VIX_PROPERTY_NONE);

// Handle errors and release handles
```

It's true that the deeper I go in this chapter the more functionality I place into code comments. The fact is you don't need me to demonstrate how to create a job handle, host handle, or virtual machine handle.

You've already done that. Like many functions in this release most of the options are not implemented or coded with a single option. That will change as the library matures. For this one provide the virtual machine handle and default the rest of the options. This will result in an asynchronous operation that will create a new snapshot of your virtual machine. I should note again that the virtual machine can be powered on or off for this to work; it doesn't matter. Remember that in this release you can only have a single snapshot. If one already exists, it will simply be replaced. Snapshot functionality is severely limited in this release of Vix. Use it only when you fully understand what your options truly are.

VixVM_GetNumRootSnapshots

This function returns the number of root snapshots contained in the referenced virtual machine. Some caution must be taken here to support virtual machines created by other products like VMware Workstation. Because the Vix API supports only a single snapshot per virtual machine if you import and use a virtual machine created with VMware Workstation, which supports multiple snapshots and multiple roots, Vix will return only a single root snapshot and ignore all others. And because other products like VMware Workstation can not only create many snapshots but also create various branches of snapshots resulting in multiple root snapshots, using the Vix API to manage multiple snapshots is a bad idea at this time. I recommend you avoid the snapshot functions until a later release of the Vix API. The function prototype is provided in the following code block:

```
VixError VixVM_GetNumRootSnapshots(VixHandle vmHandle,
                                   int *result);
```

Parameters

Parameter	Description
`vmHandle`	The handle that represents the virtual machine you want to retrieve a snapshot count for. You create this handle by calling the `VixVM_Open()` function.
`result`	The number of root snapshots present. At this time the Vix API will only return either a 0 or 1.

Returns

This function is synchronous and, therefore, simply returns an error code of type `VixError`.

Example

```
VixError err;
VixHandle jobHandle = VIX_INVALID_HANDLE;
VixHandle hostHandle = VIX_INVALID_HANDLE;
VixHandle vmHandle = VIX_INVALID_HANDLE;
VixHandle snapshotHandle = VIX_INVALID_HANDLE;
int* result;

// Use VixHost_Connect and the jobHandle to retrieve a valid hostHandle

// Use VixVM_Open to connect to a virtual machine and retrieve a vmHandle

// The virtual machine can be powered on or off at this point.
```

```
err = VixVM_GetNumRootSnapshots(vmHandle,
                                &result);

// Handle errors and release handles
```

Again, I discourage you from manipulating snapshots with this release of the Vix API; however in the interest of being thorough, this example is rather straightforward. This function is synchronous, which differs from the past several functions. That means it simply returns an error code of type `VixError`. Provide the function with a valid virtual machine handle, and it will place either a zero (0) or a one (1) into `result`.

VixVM_GetRootSnapshot

This function allows you to get a snapshot handle that represents a current snapshot on the referenced virtual machine. Please read the warning mentioned under the `VixVM_GetNumRootSnapshots()` earlier in the chapter before using Vix to handle your virtual machine snapshots. The current API implementation is very limited for snapshot management. The function prototype is provided in the following code block:

```
VixError VixVM_GetRootSnapshot(VixHandle vmHandle,
                               int index,
                               VixHandle *snapshotHandle);
```

Parameters

Parameter	Description
`vmHandle`	The handle that represents the virtual machine you want to retrieve the root snapshot for. You create this handle by calling the `VixVM_Open()` function.
`index`	The index of the root snapshot. Currently this version of the API will return only a 0.
`snapshotHandle`	The handle that represents the snapshot at the index specified.

Returns

This function is synchronous and, therefore, simply returns an error code of type `VixError`.

Example

```
VixError err;
VixHandle jobHandle = VIX_INVALID_HANDLE;
VixHandle hostHandle = VIX_INVALID_HANDLE;
VixHandle vmHandle = VIX_INVALID_HANDLE;
int index = 0;

// Use VixHost_Connect and the jobHandle to retrieve a valid hostHandle

// Use VixVM_Open to connect to a virtual machine and retrieve a vmHandle
```

```
// The virtual machine can be powered on or off at this point.

err = VixVM_GetRootSnapshot(vmHandle,
                            &index,
                            &snapshotHandle);

// Handle errors and release handles
```

This function becomes a little more important when combined with the next two, `VixVM_RemoveSnapshot()` and `VixVM_RevertToSnapshot()`. Provide a valid virtual machine handle and the root snapshot number, which in this release can only be zero (0), and it will provide a snapshot handle for the root snapshot. Other than creating a new snapshot, it is the only way to get a valid snapshot handle.

VixVM_RemoveSnapshot

This function removes a previously created snapshot referenced by the snapshot handle. To remove a snapshot the virtual machine must be powered off. The function prototype is provided in the following code block:

```
VixHandle VixVM_RemoveSnapshot(VixHandle vmHandle,
                               VixHandle snapshotHandle,
                               int options,
                               VixEventProc *callbackProc,
                               void *clientData);
```

Parameters

Parameter	Description
`vmHandle`	The handle that represents the virtual machine you want to delete a snapshot for. You create this handle by calling the `VixVM_Open()` function.
`snapshotHandle`	The handle that represents the snapshot you want to remove. You create this handle by calling the `VixVM_GetRootSnapshot()` function.
`options`	Must be zero (0) in this release.
`callbackProc`	A callback function that's called for the function's events.
`clientData`	User-defined data passed to the callback function.

Returns

This is an asynchronous call and returns a job handle.

Example

```
VixError err;
VixHandle jobHandle = VIX_INVALID_HANDLE;
VixHandle hostHandle = VIX_INVALID_HANDLE;
VixHandle vmHandle = VIX_INVALID_HANDLE;
```

```
VixHandle snapshotHandle = VIX_INVALID_HANDLE;
int numRootSnapshots;

// Use VixHost_Connect and the jobHandle to retrieve a valid hostHandle

// Use VixVM_Open to connect to a virtual machine and retrieve a vmHandle

// The virtual machine must be powered off at this point.

// Get the number of root snapshots.  Currently this will only return 0 or 1.
err = VixVM_GetNumRootSnapshots(vmHandle,
                                &numRootSnapshots);

//Handle any errors

// Now, grab a handle for the snapshot.  Remember, there can be only 1.
err = VixVM_GetRootSnapshot(vmHandle,
                            0, \\ Index must always be 0 in this release
                            &snapshotHandle);

//Handle any errors

jobHandle = VixVM_RemoveSnapshot(vmHandle,
                                 snapshotHandle,
                                 0,
                                 NULL,
                                 NULL);

err = VixJob_Wait(jobHandle, VIX_PROPERTY_NONE);

// Handle any errors and release handles
```

I left off a bit of the preceding snapshot function in this one to illustrate a point. In order to get a valid snapshot handle, you have to at least use the `VixVM_GetRootSnapshot()` function. It's hard to illustrate how this will work in future releases that support more than one snapshot, but the idea will most likely revolve around looping through the number of snapshots provided by the `VixVM_GetNumRootSnapshots()` function. In this case, because you are limited to only one, that is not necessary. Provide the `VixVM_RemoveSnapshot()` function with a valid virtual machine handle, a valid snapshot handle, and options of zero (0). Once the asynchronous operation completes the snapshot will be removed permanently.

VixVm_RevertToSnapshot

This is the final snapshot function. It takes a snapshot handle and reverts the virtual machine back to the state present in the snapshot. The virtual machine must be powered off during this operation, and depending upon the power state it was in when the snapshot was taken, it may power itself on when the snapshot is restored. The function prototype is provided in the following code block:

```
VixHandle VixVM_RevertToSnapshot(VixHandle vmHandle,
                                 VixHandle snapshotHandle,
                                 int options,
```

```
                                    VixHandle propertyListHandle,
                                    VixEventProc *callbackProc,
                                    void *clientData);
```

Parameters

Parameter	Description
`vmHandle`	The handle that represents the virtual machine you want to revert the snapshot in. You create this handle by calling the `VixVM_Open()` function.
`snapshotHandle`	The handle that represents the snapshot you want to revert to. You create this handle by calling the `VixVM_GetRootSnapshot()` function.
`options`	Must be zero (0) in this release.
`propertyListHandle`	Must be `VIX_INVALID_HANDLE` in this release.
`callbackProc`	A callback function that's called for the function's events.
`clientData`	User-defined data that is passed to the callback function.

Returns

This is an asynchronous call and returns a job handle.

Example

```
VixError err;
VixHandle jobHandle = VIX_INVALID_HANDLE;
VixHandle hostHandle = VIX_INVALID_HANDLE;
VixHandle vmHandle = VIX_INVALID_HANDLE;
VixHandle snapshotHandle = VIX_INVALID_HANDLE;

int snapshotIndex = 0; \\Only one snapshot supported so the index is always 0.

// Use VixHost_Connect and the jobHandle to retrieve a valid hostHandle

// Use VixVM_Open to connect to a virtual machine and retrieve a vmHandle

// The virtual machine must be powered off at this point.

// Now, grab a handle for the snapshot using VixVM_GetRootSnapshot.
err = VixVM_GetRootSnapshot(vmHandle,
                            snapshotIndex,
                            &snapshotHandle);

//Handle any errors

// Revert to the snapshot handle
jobHandle = VixVM_RevertToSnapshot(vmHandle,
                                   snapshotHandle,
                                   0,
```

```
                                    VIX_INVALID_HANDLE,
                                    NULL,
                                    NULL);

err = VixJob_Wait(jobHandle, VIX_PROPERTY_NONE);

// Handle any errors and release handles
```

Again, you first need a valid snapshot handle. To get this you use the `VixVM_GetRootSnapshot()` function again. Once you have it, pass it along with the virtual machine handle to the function to revert the virtual machine to that snapshot. The virtual machine must be powered off for this function to complete successfully. If the snapshot was taken while the virtual machine was powered on, it will transition into that state before the operation completes.

Now you start getting into the interesting stuff. The rest of the virtual machine operations leave the COM and Perl APIs in the dust. You are in for a treat; however, remember that this is version 1 of the Vix API, so hopefully some more flexibility will come later.

VixVM_InstallTools

This function installs the current version of VMware Tools on the guest machine. The virtual machine must be powered on for this to work. If VMware Tools is already installed, it will either upgrade it to the current version or do nothing. The function prototype is provided in the following code block:

```
VixHandle VixVM_InstallTools(VixHandle vmHandle,
                             int options,
                             char *commandLineArgs,
                             VixEventProc *callbackProc,
                             void *clientData);
```

Parameters

Parameter	Description
`vmHandle`	The handle that represents the virtual machine you want to install VMware Tools on. You create this handle by calling the `VixVM_Open()` function.
`options`	Must be zero (0) in this release.
`commandLineArgs`	Must be NULL in this release.
`callbackProc`	A callback function that's called for the function's events.
`clientData`	User-defined data to be passed to the callback function.

Returns

This is an asynchronous call and returns a job handle.

Example

```
VixError err;
VixHandle jobHandle = VIX_INVALID_HANDLE;
VixHandle hostHandle = VIX_INVALID_HANDLE;
VixHandle vmHandle = VIX_INVALID_HANDLE;

// Use VixHost_Connect and the jobHandle to retrieve a valid hostHandle

// Use VixVM_Open to connect to a virtual machine and retrieve a vmHandle

// Make sure the vm is powered on

//Install VMware Tools on the virtual machine.
jobHandle = VixVM_InstallTools(vmHandle,
                               0,
                               NULL,
                               NULL,
                               NULL);

err = VixJob_Wait(jobHandle, VIX_PROPERTY_NONE);

// Handle any errors and release handles
```

This is really simple but powerful. This is where the Perl and COM APIs are left in the dust. In this example you obtain a valid virtual machine handle and pass it to the `VixVM_InstallTools()` function. For this release of Vix `options` and `commandLineArgs` are zero (0) and `NULL`, respectively. If VMware Tools is already installed and at the current version, it will simply do nothing. If not, it will install or upgrade the package.

VixVM_WaitForToolsInGuest

This function allows you to poll the guest machine to tell if the VMware Tools package is up and running. By default a virtual machine handle has a property called `VIX_PROPERTY_VM_TOOLS_STATE` that indicates the status of VMware Tools. If they are not currently running, this function provides a wait for a specified time and updates the `vmHandle` property once that status is received. The function prototype is provided in the following code block:

```
VixHandle VixVM_WaitForToolsInGuest(VixHandle vmHandle,
                                    int timeoutInSeconds,
                                    VixEventProc *callbackProc,
                                    void *clientData);
```

Parameters

Parameter	Description
`vmHandle`	The handle that represents the virtual machine you want to check the status of VMware Tools on. You create this handle by calling the `VixVM_Open()` function.
`timeoutInSeconds`	The timeout in seconds the job waits until it reports an error.

Table continued on following page

Parameter	Description
`callbackProc`	A callback function that's called for the function's events.
`clientData`	User-defined data that is passed to the callback function.

Returns

This is an asynchronous call and returns a job handle.

Example

```
VixError err;
VixHandle jobHandle = VIX_INVALID_HANDLE;
VixHandle hostHandle = VIX_INVALID_HANDLE;
VixHandle vmHandle = VIX_INVALID_HANDLE;

int toolsActive;

// Use VixHost_Connect and the jobHandle to retrieve a valid hostHandle

// Use VixVM_Open to connect to a virtual machine and retrieve a vmHandle

// Make sure the vm is powered on or turn it on yourself

// Check the properties for the virtual machine for the VMware Tools status
jobHandle = VixGetProperties(vmHandle,
                             VIX_PROPERTY_VM_TOOLS_STATE,
                             toolsActive,
                             VIX_PROPERTY_NONE);

  err = VixJob_Wait(jobHandle, VIX_PROPERTY_NONE);

  if (VIX_FAILD(err)) {
    printf("\n An error occurred retrieving the VMware Tools status. \n");
    goto abort;
  }

Vix_ReleaseHandle(JobHandle);

// If the jobHandle above didn't show VMware Tools as running wait until it does
if (toolsActive != VIX_TOOLSSTATE_RUNNING) {
  jobHandle = VixVM_WaitForToolsInGuest(vmHandle,
                                        60,
                                        NULL,
                                        NULL);

err = VixJob_Wait(jobHandle, VIX_PROPERTY_NONE);

// Handle any errors and release handles
```

Again, this one is pretty cool. All virtual machine handles have a property `VIX_PROPERTY_VM_TOOLS_STATE`. By polling this property you can determine if VMware Tools has started up and in combination

with the `VixVM_WaitForToolsInGuest()` function wait until it happens. Once it does, the function will automatically update the property on the virtual machine handle to reflect the new state. Pretty slick.

VixVM_UpgradeVirtualHardware

This function checks the current version of the virtual hardware for a given virtual machine and updates it to the most current version if necessary. The virtual machine must be in a power off state to call this function. The function prototype is provided in the following code block:

```
VixHandle VixVM_UpgradeVirtualHardware(VixHandle vmHandle,
                                       int options,
                                       VixEventProc *callbackProc,
                                       void *clientData);
```

Parameters

Parameter	Description
`vmHandle`	The handle that represents the virtual machine you want to upgrade the virtual hardware on. You create this handle by calling the `VixVM_Open()` function.
`options`	Must be zero (0) in this release.
`callbackProc`	A callback function that's called for the function's events.
`clientData`	User-defined data that is passed to the callback function.

Returns

This is an asynchronous call and returns a job handle.

Example

```
VixError err;
VixHandle jobHandle = VIX_INVALID_HANDLE;
VixHandle hostHandle = VIX_INVALID_HANDLE;
VixHandle vmHandle = VIX_INVALID_HANDLE;

// Use VixHost_Connect and the jobHandle to retrieve a valid hostHandle

// Use VixVM_Open to connect to a virtual machine and retrieve a vmHandle

// Make sure the vm is powered off

jobHandle = VixVM_UpgradeVirtualHardware(vmHandle,
                                         0,
                                         NULL,
                                         NULL);

err = VixJob_Wait(jobHandle,
                  VIX_PROPERTY_NONE);

// Handle any errors and release handles
```

Not much to this one. This is exactly like selecting the Upgrade Virtual Hardware selection from the menu in the VMware Server Console. The only time you will need to do this is if you use a virtual machine created on an older version of VMware. The console will let you know if this needs to be done, so don't worry about it unless that happens. Like before most of the options are not implemented, so provide a valid virtual machine handle, provide zero (0) to options, and unless using a callback `NULL` them out. Once the operation completes, the hardware is upgraded.

VixVM_LoginInGuest

This function is really slick. It solves a problem you will see in the next two chapters that has to do with accessing the virtual machine via authentication through Perl. The virtual machine must be powered on and VMware Tools running before this function will work. The idea here is that through the use of VMware Tools this function can authenticate and login via code. This idea is great in terms of squeezing everything you can out of your scripts. This provides the necessary authentication information for the next several functions. The function prototype is provided in the following code block:

```
VixHandle VixVM_LoginInGuest(VixHandle vmHandle,
                             char *userName,
                             char *password,
                             int options,
                             VixEventProc *callbackProc,
                             void *clientData);
```

Parameters

Parameter	Description
`vmHandle`	The handle that represents the virtual machine you want to log in to. You create this handle by calling the `VixVM_Open()` function.
`userName`	The user name of an account of the guest machine.
`password`	The password that corresponds to the user name specified previously.
`options`	Must be zero (0) in this release.
`callbackProc`	A callback function that's called for the function's events.
`clientData`	User-defined data that is passed to the callback function.

Returns

This is an asynchronous call and returns a job handle.

Example

```
VixError err;
VixHandle jobHandle = VIX_INVALID_HANDLE;
VixHandle hostHandle = VIX_INVALID_HANDLE;
VixHandle vmHandle = VIX_INVALID_HANDLE;

// Use VixHost_Connect and the jobHandle to retrieve a valid hostHandle

// Use VixVM_Open to connect to a virtual machine and retrieve a vmHandle
```

```
// Make sure the vm is powered on or turn it on yourself

// Wait until VMware Tools is up and running.
jobHandle = VixVM_WaitForToolsInGuest(vmHandle,
                                      120,
                                      NULL,
                                      NULL);

err = VixJob_Wait(jobHandle, VIX_PROPERTY_NONE);

// Handle any errors

// Login to the guest
jobHandle = VixVM_LoginInGuest(vmHandle,
                               "administrator",
                               "pass@word1",
                               0,
                               NULL,
                               NULL);

err = VixJob_Wait(jobHandle, VIX_PROPERTY_NONE);

// Handle any errors and release handles
```

VMware Tools is required for this to work, so you'll notice that after I get the virtual machine handle I wait until VMware Tools is running before calling the `VixVM_LoginInGuest()` function. This is required; you must wait until VMware Tools is running or it will fail. Pass the login credentials to the `VixVM_LoginToGuest()` function and off you go. There's little you can do with this, however, until you combine it with the next several functions. Together they are a powerful addition to the API line.

VixVM_RunProgramInGuest

Continuing on the useful function path, this one executes a program on the guest machine. You must call `VixVM_LoginInGuest()` before you call this function. The function prototype is provided in the following code block:

```
VixHandle VixVM_RunProgramInGuest(VixHandle vmHandle,
                                  const char *guestProgramName,
                                  const char *commandLineArgs,
                                  VixRunProgramOptions options,
                                  VixHandle propertyListHandle,
                                  VixEventProc *callbackProc,
                                  void *clientData);
```

Parameters

Parameter	Description
`vmHandle`	The handle that represents the virtual machine you want to run the application on. You create this handle by calling the `VixVM_Open()` function.

Table continued on following page

Parameter	Description
guestProgramName	The path and program name of the executable you want to run on the guest machine.
commandLineArgs	Any command-line arguments you want to pass to the executable mentioned previously.
options	Run options for the program. Currently the API supports two options. The first is zero (0), and this causes the function to wait until the application exists on the guest operating system. The second is VIX_RUNPROGRAM_RETURN_IMMEDIATELY, and this causes the function to return as soon as the guest program starts.
propertyListHandle	This must be VIX_INVALID_HANDLE in this release.
callbackProc	A callback function that's called for the function's events.
clientData	User-defined data that is passed to the callback function.

Returns

This is an asynchronous call and returns a job handle.

Example

```
VixError err;
VixHandle jobHandle = VIX_INVALID_HANDLE;
VixHandle hostHandle = VIX_INVALID_HANDLE;
VixHandle vmHandle = VIX_INVALID_HANDLE;

// Use VixHost_Connect and the jobHandle to retrieve a valid hostHandle

// Use VixVM_Open to connect to a virtual machine and retrieve a vmHandle

// Make sure the vm is powered on or turn it on yourself

// Wait until VMware Tools is up and running.

// Login to the guest

// Run an application in the guest
jobHandle = VixVM_RunProgramInGuest(vmHandle,
                                    "C:\\WINDOWS\\SYSTEM32\\CALC.exe",
                                    NULL,
                                    VIX_RUNPROGRAM_RETURN_IMMEDIATELY,
                                    VIX_INVALID_HANDLE,
                                    NULL,
                                    NULL);

err = VixJob_Wait(vmHandle, VIX_PROPERTY_NONE);

// Handle any errors and release handles.
```

Here's where the power of this new API takes form. In this example I demonstrate how you can log in to the guest using the previous `VixVM_LoginToGuest()` function and then run a program on the guest machine. This breathes new life into the automation capabilities at your disposal.

VixVM_CopyFileFromHostToGuest

These last two virtual machine operations make the Vix API worth using if nothing else does. This first function copies the file specified from the host machine to the guest machine. You must call `VixVM_LoginInGuest()` before you call this function. The function prototype is provided in the following code block:

```
VixHandle VixVM_CopyFileFromHostToGuest(VixHandle vmHandle,
                                        const char *hostPathName,
                                        const char *guestPathName,
                                        int options,
                                        VixHandle propertyListHandle,
                                        VixEventProc *callbackProc,
                                        void *clientData);
```

Parameters

Parameter	Description
`vmHandle`	The handle that represents the virtual machine you want to copy files to. You create this handle by calling the `VixVM_Open()` function.
`hostPathName`	The name of a file on the host machine to copy.
`guestPathName`	The location on the guest machine to copy the file to.
`options`	Must be zero (0) in this release.
`propertyList`	Must be `VIX_INVALID_HANDLE` in this release.
`callbackProc`	A callback function that's called for the function's events.
`clientData`	User-defined data that is passed to the callback function.

One caveat for this function — when it refers to the host, it is talking about the machine that the script is run from, not the host server as it leads you to believe. This may or may not be the intended behavior; however, it is true at the present time.

Returns

This is an asynchronous call and returns a job handle.

Example

```
VixError err;
VixHandle jobHandle = VIX_INVALID_HANDLE;
VixHandle hostHandle = VIX_INVALID_HANDLE;
VixHandle vmHandle = VIX_INVALID_HANDLE;

// Use VixHost_Connect and the jobHandle to retrieve a valid hostHandle
```

```
// Use VixVM_Open to connect to a virtual machine and retrieve a vmHandle

// Make sure the vm is powered on or turn it on yourself

// Wait until VMware Tools is up and running.

// Login to the guest

// Copy files from the host to the guest
jobHandle = VixVM_CopyFileFromHostToGuest(vmHandle,
                                          "C:\\location of file",
                                          "C:\\location to copy to",
                                          0,
                                          VIX_INVALID_HANDLE,
                                          NULL,
                                          NULL);

err = VixJob_Wait(jobHandle, VIX_PROPERTY_NONE);

// Handle any errors and release handles.
```

Now Perl and COM are starting to look like child's play. Here, by utilizing the `VixVM_LoginToGuest()` function along with `VixVM_CopyFileFromHostToGuest()` you can do some amazing things. This example shows how you can log in to the guest and copy files from the host to the guest machine. In the coming chapters you will see me using Perl and COM to provide real-world examples of pushing builds out to testing machines. Unfortunately, due to how new the Vix API is, I didn't feel as though it was appropriate to use it in those examples. Because of that you'll see me do some real back bending to get files from the host machine up to the guest. As you can tell that is soon to be a thing of the past. Hopefully, once Vix hits generation two, it will become your only stop for API needs. There are, however, still some bumps in the road that need to be addressed before you can use it for all your needs.

VixVM_CopyFileFromGuestToHost

This is just the inverse of the preceding function. Here you copy a file from a location on the guest machine to a location on the host server. Like before, you must call `VixVM_LoginInGuest()` before you call this function. The function prototype is provided in the following code block:

```
VixHandle VixVM_CopyFileFromGuestToHost(VixHandle vmHandle,
                                        const char *guestPathName,
                                        const char *hostPathName,
                                        int options,
                                        VixHandle propertyListHandle,
                                        VixEventProc *callbackProc,
                                        void *clientData);
```

Parameters

Parameter	Description
`vmHandle`	The handle that represents the virtual machine you want to copy files from. You create this handle by calling the `VixVM_Open()` function.

Parameter	Description
`guestPathName`	The name of the file on the guest machine to copy.
`hostPathName`	The location on the host machine to copy the file to.
`options`	Must be zero (0) in this release.
`propertyList`	Must be `VIX_INVALID_HANDLE` in this release.
`callbackProc`	A callback function that's called for the function's events.
`clientData`	User-defined data that is passed to the callback function.

One caveat for this function — when it refers to the host, it is talking about the machine that the script is run from, not the host server as it leads you to believe. This may or may not be the intended behavior; however, it is true at the present time.

Returns

This is an asynchronous call and returns a job handle.

Example

```
VixError err;
VixHandle jobHandle = VIX_INVALID_HANDLE;
VixHandle hostHandle = VIX_INVALID_HANDLE;
VixHandle vmHandle = VIX_INVALID_HANDLE;

// Use VixHost_Connect and the jobHandle to retrieve a valid hostHandle

// Use VixVM_Open to connect to a virtual machine and retrieve a vmHandle

// Make sure the vm is powered on or turn it on yourself

// Wait until VMware Tools is up and running.

// Login to the guest

// Copy files from the guest to the host
jobHandle = VixVM_CopyFileFromGuestToHost(vmHandle,
                                          "C:\\location of file",
                                          "C:\\location to copy to",
                                          0,
                                          VIX_INVALID_HANDLE,
                                          NULL,
                                          NULL);

err = VixJob_Wait(jobHandle, VIX_PROPERTY_NONE);

// Handle any errors and release handles.
```

This is, of course, the opposite of the previous function. Here you can copy a file from the guest machine to the host machine. Pretty simple.

Job Handle Operations

The following functions are used against job handles themselves. Some of the functions like `VixJob_Wait()` you've already seen, but I want to review the three types nevertheless.

> *Unless otherwise noted the examples provided are only to demonstrate usage and will not compile on their own.*

VixJob_Wait

This function is a blocking function that waits for the specified job handle to complete its operation. This function also allows for the passing of various properties that the job handle possesses once it completes its operation. The function prototype is provided in the following code block:

```
VixError VixJob_Wait(VixHandle jobHandle,
                     VixPropertyID propertyID,
                     ...,
                     VIX_PROPERTY_NONE);
```

Parameters

Parameter	Description
`jobHandle`	A job handle from any asynchronous Vix function.
`propertyID`	The name of the name/value pair for the property you wish to retrieve. The list must end with the special `VIX_PROPERTY_NONE` property.

Returns

This function is synchronous, so returns an error code of type `VixError`.

> *See the section "Blocking until the Job Is Complete" earlier in the chapter for an example of this function.*

VixJob_GetError

This function is really no different than the error code that is already provided by the `VixJob_Wait()` function. The difference here is that you do not grab the `VixError` object until after the `VixJob_Wait()` function completes. I provide an example following the parameters and returns to illustrate my point. The function prototype is provided in the following code block:

```
VixError VixJob_GetError(VixHandle jobHandle);
```

Parameters

Parameter	Description
`jobHandle`	A job handle from any asynchronous Vix function.

Returns

This function is synchronous, so it returns an error code of type `VixError`.

Example

```
// Retrieve error with VixJob_Wait

err = VixJob_Wait(jobHandle, VIX_PROPERTY_NONE);



// Retrieve same error with VixJob_GetError

VixJob_Wait(jobHandle, VIX_PROPERTY_NONE);

err = VixJob_GetError(jobHandle);
```

VixJob_CheckCompletion

You saw this function earlier in the chapter. Essentially what this function does is allow you to poll the status of a job handle for completion. The function prototype is provided in the following code block:

```
VixError VixJob_CheckCompletion(VixHandle jobHandle,
                                Bool *complete);
```

Parameters

Parameter	Description
`jobHandle`	A job handle from any asynchronous Vix function.
`Complete`	A Boolean value that indicates if the job is complete or not.

Returns

This function is synchronous, so it returns an error code of type `VixError`.

See the section "Polling for Job Completion" earlier in the chapter for an example of this function.

Using Vix — A Few Brief Examples

Now that I've gone through the various functionality of Vix and discussed some of its stronger and weaker points, I want to show an example that wraps up some of the more common functions used in Vix into something a bit more practical.

First you can whip up a quick executable that allows you to remotely register a virtual machine. Using the setup procedure at the beginning of this chapter, create a project named `VixRegisterVM`. This executable will take arguments from the command line in the following format:

```
VixRegisterVM.exe <host> <username> <password> <vmx path>
```

Visual C++ 2005 defaults to using the Unicode character set. To avoid any character conversions for the command-line arguments, I switched the character set project defaults to Not Set. This can be done through the project properties, Configuration Properties➪General➪Character Set.

I'm not going to provide any error checking of the arguments nor any command-line help. I just want to keep it simple for now.

Enter the following into your `VixRegisterVM.cpp` code file in Visual Studio, replacing what's already in the file:

```
#include "stdafx.h"
#include "vix.h"

int _tmain(int argc, char *argv[])
{
  VixHandle hostHandle = VIX_INVALID_HANDLE;
  VixHandle jobHandle = VIX_INVALID_HANDLE;
  VixError err = VIX_OK;

  // Connect to the server with the username and
  // password specified on the command line
  jobHandle = VixHost_Connect(VIX_API_VERSION,
                              VIX_SERVICEPROVIDER_VMWARE_SERVER,
                              argv[1],
                              0,
                              argv[2],
                              argv[3],
                              0,
                              VIX_INVALID_HANDLE,
                              NULL,
                              NULL);

  // Use a blocking function to wait for the job to complete.
  // Then retrieve the hostHandle from the job.
  err = VixJob_Wait(jobHandle,
                    VIX_PROPERTY_JOB_RESULT_HANDLE,
                    &hostHandle,
                    VIX_PROPERTY_NONE);

  if (VIX_FAILED(err)) {
    printf("\n An error occurred connecting to the specified host. \n");
    goto abort;
  }

  Vix_ReleaseHandle(jobHandle);

  // Now that we have a connection to the host let's register the virtual machine.
```

```
    jobHandle = VixHost_RegisterVM(hostHandle,
                                   argv[4],
                                   NULL,
                                   NULL);

    err = VixJob_Wait(jobHandle,
                      VIX_PROPERTY_NONE);

    if (VIX_FAILED(err)) {
      printf("\n An error occurred registering the specified host. \n");
      goto abort;
    }

    printf("\n The virtual machine has been registered. \n");

    Vix_ReleaseHandle(jobHandle);
    Vix_ReleaseHandle(hostHandle);
    return 0;

  abort:
    Vix_ReleaseHandle(jobHandle);
    Vix_ReleaseHandle(hostHandle);
    return -1;
  }
```

This is a pretty simple example but it works. Here you accept the arguments of `<host>`, `<username>`, `<password>`, and `<vmx path>` from the command line and pass them into the appropriate functions. Technically `argv[]`, where the command-line arguments are exposed to the code, is zero based; however, the first element of that array always contains the program name, so remember to start with element one.

Now, you can get just a wee bit more complicated and show off just one of the fancy features of Vix. Here you build upon the previous example by connecting to the virtual machine, powering it on, and copying a file up to it. The command-line arguments are as follows:

```
VixCopyToGuest.exe <host> <username> <password> <vmx path>
```

Start up a new project using the guidelines provided at the beginning of this chapter and name it `VixCopyToGuest`. Again, like the previous example, set this project's character set to `Not Set` so you can utilize the command-line arguments with little effort.

Once the project is up open the `VixCopyToGuest.cpp` code file and enter the following, replacing what's already in the file:

```
#include "stdafx.h"
#include "vix.h"

int _tmain(int argc, char *argv[])
{
  VixHandle hostHandle = VIX_INVALID_HANDLE;
  VixHandle jobHandle = VIX_INVALID_HANDLE;
  VixHandle vmHandle = VIX_INVALID_HANDLE;
  VixError err = VIX_OK;
```

```
int toolsActive;

// Connect to the server with the username and
// password specified on the command line
jobHandle = VixHost_Connect(VIX_API_VERSION,
                            VIX_SERVICEPROVIDER_VMWARE_SERVER,
                            argv[1],
                            0,
                            argv[2],
                            argv[3],
                            0,
                            VIX_INVALID_HANDLE,
                            NULL,
                            NULL);

// Use a blocking function to wait for the job to complete.
// Then retrieve the hostHandle from the job.
err = VixJob_Wait(jobHandle,
                  VIX_PROPERTY_JOB_RESULT_HANDLE,
                  &hostHandle,
                  VIX_PROPERTY_NONE);

if (VIX_FAILED(err)) {
  printf("\n An error occurred connecting to the specified host. \n");
  goto abort;
}

Vix_ReleaseHandle(jobHandle);

// Now, let's open the virtual machine provided on the command line.
// This assumes it is already registered.
jobHandle = VixVM_Open(hostHandle,
                       argv[4],
                       NULL,
                       NULL);

err = VixJob_Wait(jobHandle,
                  VIX_PROPERTY_JOB_RESULT_HANDLE,
                  &vmHandle,
                  VIX_PROPERTY_NONE);

if (VIX_FAILED(err)) {
  printf("\n An error occurred connecting to the virtual machine. \n");
  goto abort;
}

Vix_ReleaseHandle(jobHandle);

// Now, we have our vmHandle, let's startup the virtual machine.
jobHandle = VixVM_PowerOn(vmHandle,
                          VIX_VMPOWEROP_NORMAL,
                          VIX_INVALID_HANDLE,
                          NULL,
                          NULL);
```

```
err = VixJob_Wait(jobHandle,
                  VIX_PROPERTY_NONE);

if (VIX_FAILED(err)) {
  printf("\n An error occurred powering on the virtual machine. \n");
  goto abort;
}

printf("\n The virtual machine is powering on. \n");

Vix_ReleaseHandle(jobHandle);

// Ok, while that's starting up let's start looking for the VMware Tools
// active status since we need it for the next steps.
jobHandle = Vix_GetProperties(vmHandle,
                              VIX_PROPERTY_VM_TOOLS_STATE,
                              &toolsActive,
                              VIX_PROPERTY_NONE);

err = VixJob_Wait(jobHandle,
                  VIX_PROPERTY_NONE);

// ** See the notes at the end of this code block for why I have
// this commented out. **

//if (VIX_FAILED(err)) {
//  printf("\n An error occurred retrieving the VMware Tools status. \n");
//  goto abort;
//}

Vix_ReleaseHandle(jobHandle);

// If the jobHandle above didn't show VMware Tools as running wait until it does
if (toolsActive != VIX_TOOLSSTATE_RUNNING) {
  jobHandle = VixVM_WaitForToolsInGuest(vmHandle,
                                        60,
                                        NULL,
                                        NULL);

  printf("\n Waiting for VMware Tools to come online. \n");

  err = VixJob_Wait(jobHandle, VIX_PROPERTY_NONE);

  if (VIX_FAILED(err)) {
    printf("\n The VMware Tools status could not be found or the time out ↩
            was reached. \n");
    goto abort;
  }
}

Vix_ReleaseHandle(jobHandle);

// Now, if we make it here then VMware Tools is running and ready for us
// to do some magic.  Let's login to the guest.
```

```
  jobHandle = VixVM_LoginInGuest(vmHandle,
                                 "administrator",
                                 "pass@word1",
                                 0,
                                 NULL,
                                 NULL);

  printf("\n Logging into guest. \n");

  err = VixJob_Wait(jobHandle,
                    VIX_PROPERTY_NONE);

  if (VIX_FAILED(err)) {
    printf("\n Login to guest failed. \n");
    goto abort;
  }

  Vix_ReleaseHandle(jobHandle);

  // Since we made it this far let's copy a file up to the guest machine.
  jobHandle = VixVM_CopyFileFromHostToGuest(vmHandle,
                                            "c:\\test.txt",
                                            "c:\\test.txt",
                                            0,
                                            VIX_INVALID_HANDLE,
                                            NULL,
                                            NULL);

  printf("\n Copying file. \n");

  err = VixJob_Wait(jobHandle,
                    VIX_PROPERTY_NONE);

  if (VIX_FAILED(err)) {
    printf("\n File copy failed. \n");
    goto abort;
  }

  Vix_ReleaseHandle(jobHandle);

  // So, here we are... at the bottom.  If we make it here then all went well.
  printf("\n The operation completed successfully. \n");

  Vix_ReleaseHandle(jobHandle);
  Vix_ReleaseHandle(hostHandle);
  Vix_ReleaseHandle(vmHandle);
  return 0;

abort:
  Vix_ReleaseHandle(jobHandle);
  Vix_ReleaseHandle(hostHandle);
  Vix_ReleaseHandle(vmHandle);
  return -1;
}
```

Beautiful. As long as the file you want to copy exists, this script will start up the virtual machine you specify and copy a file up to it. One thing to note, however, is that I hard-coded the user name and password that you log in to the guest with. I did this to keep the command line from being packed full of arguments; however, it would be very easy to make these arguments as well. Just follow the format I already used for the host name.

Now, if you read the preceding code carefully, you may have noticed that I commented out a section of code. Why would I do that in a book? Well, it's to prove a point. Remember at the beginning of this chapter I provided a link to the release notes for this version of the Vix API? Well, here's where it comes into play. There is a bug in v1.0 of Vix that happens when you attempt to retrieve the `VIX_PROPERTY_VM_TOOLS_STATE` property from a virtual machine handle. The normal flow, and how you should do it under better circumstances, would be to check the error status of the `VixVM_GetProperties()` function before moving on. This, however, will break this script every time by throwing an invalid argument error. To get around it I simply commented out the error check for that section of code. I left it in, however, to illustrate that it should be there and that this API is still a little buggy.

Over the next two chapters you see some real-world examples where I use the Perl and COM APIs to perform some more advanced tasks. I want to stress, however, that Perl and COM are not your only options. The second example I provided in this section clearly shows that Vix can accomplish the goals I set forth to do in Chapters 11 and 12 with less effort. This example also shows you why I chose not to use Vix in the real-world examples. It is still a bit new but possesses great potential.

Summary

With the introduction of VMware Server came the Programming API, or Vix. As you turn away from this chapter I want you to remember a couple key points:

- Vix possesses a great deal of functionality, not the least of which is the ability to authenticate and copy files to and from guest machines, which is something the other APIs can't do.
- Vix is still first generation, which somewhat limits its usefulness. One of the major limitations is the ability to handle multiple snapshots. Still, while it is new and still needs some work, it is the choice for future programming against VMware Server.

I will continue to use Perl and COM as the APIs of choice for the remainder of this book because they are time tested and work. There should be little doubt, however, that Vix is in fact the replacement for the Perl and COM APIs down the road.

11

Integrating with Your Existing Environment

Integration. A term often used loosely by management and feared by developers and system engineers. Even worse is the dreaded *seamless integration*. You and I both know seamless is usually a pipe dream. That being said it's a fact of our careers. As long as there are multiple software vendors, and there always will be, we will continue to integrate hardware and software into what we hope turns into a useful and somewhat seamless result.

The development tools both you and your team utilize right now are built upon the idea of allowing integration. Some of this, like source control, comes already integrated into your environment. Other tools require a bit of persuasion.

So, I think it's only fitting that we crank out some integration of our own. Because we are so conveniently on the topic of VMware Server and the development environment, I want to discuss integrating the power of virtual machines into your development environment. That's what this chapter is about.

> *Because the available platforms and packages are so widespread, I will be limiting this chapter to what I believe are two of the more useful and widely used packages at this time, the new Visual Studio Team System and CruiseControl.NET. The idea of VMware Server integration, however, is not for Windows-based systems alone. Although I jump into two Windows-based examples in this chapter, I don't want to leave you with the impression that this is a Windows-only idea or desire. Linux users out there can easily integrate VMware Server into their development process. I cover some points toward the end of this chapter on Linux integration tasks.*

Continuous Integration

So just what is *continuous integration*? As you develop software, the chances of just leveraging one piece of software are slim. In this day and age where things like test-driven development, unit testing, build management, and so forth become more a part of the development process, the need to integrate these tools and practices into a smooth workflow becomes paramount. This integration doesn't happen magically of course. It requires a set of tools and procedures that continue the process again and again. Unit testing is only useful if you actually run these tools. Alone they serve no real purpose other than to take up space in your menus if something isn't done to ensure their usage. As long as it is up to the end user to utilize the integrated tools their usage will remain sporadic at best. What you need to be able to do is continuously integrate the tools.

One of the best examples of how to do this in my opinion can be seen in a product called CruiseControl.NET or CCNet for short. The main idea behind CCNet is to monitor the developer check-ins to source control and, once detected, check-out the code, build it, and run a set of preconfigured tools. The results of this are returned back to the developer quickly and can be used to gauge how well his or her changes integrated into the entire code base.

Along with developer notification CCNet provides a nice web interface that displays status and reports on the builds and testing that occurs. This process is highly configurable and can support many of the widely used tools today like JUnit, NUnit, NAnt, and more. The result is a continuously integrated environment that extends from the developer's check-in through build and unit testing. All of this is handled automatically.

Another tool worth mentioning is Visual Studio Team System, or VSTS. Though still rather new, VSTS offers tight integration into Visual Studio 2005 and, if that is your development environment of choice, is worth a look. While CCNet wins when it comes to focused delivery of continuous integration, VSTS is not far behind. Through the use of scheduled builds and scripting against the new Microsoft Build tool you can trigger external tools, like NUnit testing, with every new build. The ability to write new functionality into VSTS, such as VMware Server integration, still leaves a lot to be desired. This is due mainly to the complexity that exists when tapping into the VSTS build events. More on this later in this chapter.

So what's our goal here? Automation and integration. You've already learned that with VMware Server you can script and automate your virtual machines in a wide variety of ways. Why can't you integrate this into your continuous integrated environment just described? You can, and this chapter's focus is in doing just that.

Project Setup

I have created a quick .NET project to use for the following examples. This project is available for download from the book's web site at `www.wrox.com`.

Visual Studio Team System contains its own source control system; however, for the scenario in this chapter I will not be using native VSTS source control. Instead both the CruiseControl.NET and VSTS example projects will live in a single Visual Source Safe database. The reason for this is because the CruiseControl.NET modules for VSTS are still rather new, whereas its support for Visual SourceSafe has been around the block a few times. This will hopefully keep things simple in terms of functionality out of the box for this chapter.

VMware Tools Scripting

As you are no doubt aware by now the performance of your virtual machines is enhanced considerably by the installation of VMware Tools. Not only does it install the specific drivers necessary for the guest to interface with the virtual hardware efficiently, but it also adds a bit of scripting support to the guest.

Over the next two sections you need to be able to poll your guest machine for its IP address. Unless you set up your machines with static addresses, which is doubtful, you will need this information in order to copy files up to the guest's desktop, as you will do here shortly.

The VMware Tools application will accept anything as a script so long as it is executable. You could use Perl, but you would have to install an interpreter. You could use VBScript and leverage the Windows Scripting Host without any extra software installation, so that's a solution. You could also write your own application, say in .NET, to take care of this for you. I'll show you both. First I want to tackle the VBScript version.

Scripting VMware Tools with VBScript

This script will contain the code that places the information you need into a `get_client_info()` hash that you can later retrieve from your host script. I cover where to place this script in a little bit. For now you will get what you need entered into your new script file.

Like before when you leveraged the Windows Scripting Host (WSH), two files are needed: a *.wsf* file that tells the WSH what script to run and the script itself. Start with the *.wsf* file. Open your favorite text editor and enter the information that follows. This will be your `getIP.wsf` file.

VMware Tools Scripting in Linux

Although I'm not providing an example for a Linux-based guest machine, the idea is pretty straightforward. While Windows-based guests offer a little flexibility in terms of development of the VMware Tools scripts, Linux makes it even simpler. Because almost every Linux distribution comes with Perl installed that is the obvious choice for your Linux-based VMware Tools scripts. You can create the Perl script to accomplish this task easily.

As a quick example you could use the Linux command `ifconfig` and a regular expression in Perl to extract the IP address of the guest:

```
my ($myip, @ipaddress);

$_ = `ifconfig`;

@ipaddress = /inet addr:(\d+\.\d+\.\d+\.\d+)/ig;

$myip = $ipaddress[0];
```

Once the regular expression extracts the IP address and it is placed into the `$myip` scalar, you can make an external call to the VMware Service command to set the guest variable as shown in the following code block:

```
system("/etc/vmware/vmware-guestd --cmd 'info-set guestinfo.ip $myip'");
```

The following scripts are written and performed on a Windows-based guest machine with VMware Tools installed. In order to test the scripts you must run them inside a guest machine itself.

getIP.wsf

```
<job id="GetIP">
  <script language="VBScript" src="getIP.vbs" />
</job>
```

Now that the *.wsf* file is out of the way, create a new file named `getIP.vbs` and place it into the same directory as the `getIP.wsf` file.

getIP.vbs

```
Set objShell = CreateObject("WScript.Shell")
Set objWshScriptExec = objShell.Exec("ipconfig")
Set objStdOut = objWshScriptExec.StdOut
strOutput = objStdOut.ReadAll


Set ipOnly = New RegExp
ipOnly.Pattern = "(\d+\.\d+\.\d+\.\d+)"

Set matches = ipOnly.Execute(strOutput)

objShell.Exec("VMwareService -cmd ""info-set guestinfo.ip" & matches(0) & "")
```

Though it's not required, the best place for the `getIP.wsf` *and* `getIP.vbs` *files to live is in the* `C:\Program Files\VMware\VMware Tools` *directory. If you haven't already done so, move the two files there now.*

The guts of the `getIP.vbs` file are rather simple. First, the script uses the `ipconfig` command; the script executes it and pipes its output back into the script as `strOutput`. Then, using a very simple regular expression it extracts only the IP addresses from that output as the `matches()` array. Now, assuming that this virtual machine has only one network adapter, the first address in that array should be the network interface you are looking for. Using the `VMwareService.exe` command, installed as part of the VMware Tools package, it sets a guestinfo variable called `guestinfo.ip` to the first element of the `matches` array, which should contain the guest machine's IP address.

Now, you can test it. From a command console within the directory where you saved the two preceding files, execute the following. Remember, this has to be done from within a guest machine with VMware Tools installed:

```
cscript /nologo getIP.wsf
```

If all goes well you should see absolutely nothing. That's a good thing because you don't need to see any feedback here. All you are concerned about is if the guestinfo variable `guestinto.ip` was set. You can check that now with the following:

```
"C:\Program Files\VMware\VMware Tools\VMwareService" -cmd "info-get guestinfo.ip"
```

Remember to place the path to the `VMwareService` *command in quotes if it is a path that contains spaces.*

If the preceding command returns the guest machine's IP address, then you are in business. All you have to do now is set up VMware Tools to use it.

Next, you'll notice the VMware Tools directory has several batch files in it by default. Though the VMware Tools package can utilize any executable file as a trigger script, sticking with VMware's batch file setup is what I usually do. Create a new batch file named `power-on.bat` in the VMware Tools directory and enter the following:

```
cscript /nologo getIP.wsf
```

This is the batch file that VMware Tools will call automatically on power up. This, of course, doesn't happen on its own. You need to tell VMware Tools to do this:

1. Close your open windows and locate the VMware Tools icon in the task bar of the guest now. You are looking for an icon like the one in Figure 11-1.

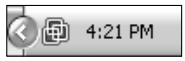

Figure 11-1

2. Double-click the icon to open up the VMware Tools application. There are several items you can configure here, but at the moment you are interested in the Scripts tab shown in Figure 11-2.

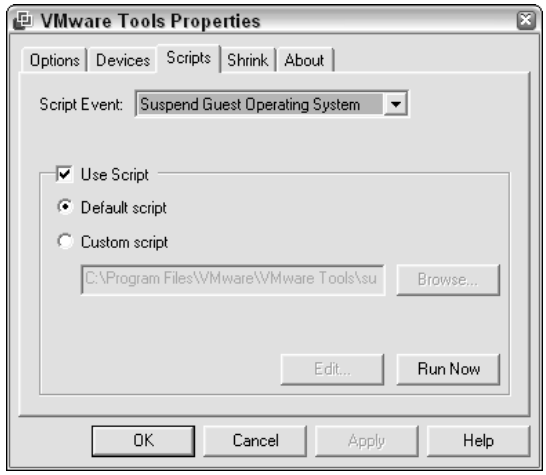

Figure 11-2

3. From the Script Event combo box drop-down select Power On Guest Operating System as the event. In the bottom portion of the window make sure Use Script is selected and choose Custom script. Click the Browse button to reveal Figure 11-3.

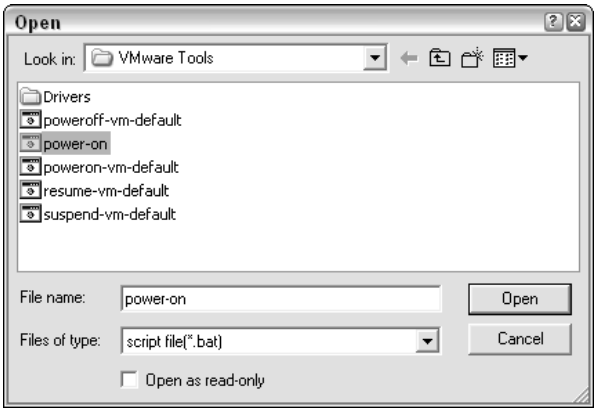

Figure 11-3

4. As you can see, the `power-on.bat` file that you created is displayed along with the default scripts VMware provided. Select the `power-on.bat` file and click OK. The screen should now match Figure 11-4.

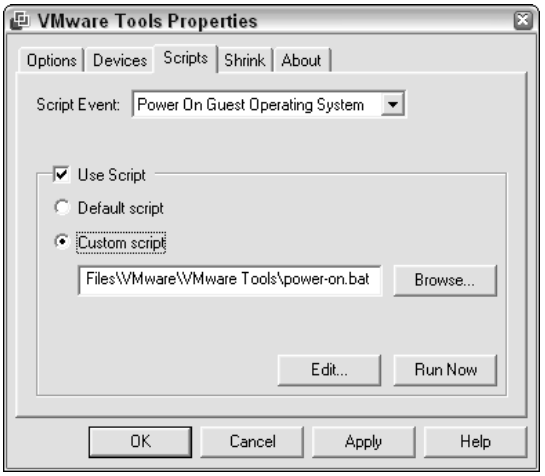

Figure 11-4

5. Click OK to exit out of the VMware Tools application.

The script you just created will live on the guest machine and get fired on every power on event by the VMware Tools package. This will write the machine's IP address to a special hash that you will use later on during the integration phase. With that first example out of the way, you can now do the same thing another way just to get a sample of the possibilities.

Scripting VMware Tools with C# and VB.NET

You are finished with the VBScript method of accomplishing this. Now you can take a look at another way that in my opinion is a little cleaner. It does, however, require you to understand how to program in C#.

First, I should tell you that this is written against the .NET Framework v2.0 so make sure it is installed on your guest machines. Chances are, as long as they are fully patched, it is.

> *The Microsoft .NET Framework is installed in the* `C:\Windows\Microsoft.NET\Framework` *directory. Each version of the framework that's installed is located in its own directory. If you see a directory similar to v2.xxxxx, then version 2 of the .NET Framework has been installed.*

Open up Visual Studio 2005 and start a new C# console application project. Name it `vmGetGuestIP`. That's catchy. Now, without code diving too much, simply copy the code that follows into the `Program.cs` file that opens automatically when you create the project. Make sure you remove everything out of the file prior to copying in this code.

Program.cs

```
using System;
using System.Collections.Generic;
using System.Text;

namespace vmGuestGetIP
{
    class Program
    {
        static void Main(string[] args)
        {
            string computerName = System.Net.Dns.GetHostName();
            string computerIP =
             System.Net.Dns.GetHostEntry(computerName).AddressList[0].ToString();
            setVMwareGuestInfo(computerIP);

        }

        static private void setVMwareGuestInfo(string computerIP)
        {
            System.Diagnostics.ProcessStartInfo proc = new
             System.Diagnostics.ProcessStartInfo("C:\\Program Files\\VMware\\VMware
             Tools\\VMwareService.exe");
            proc.Arguments = "-cmd \"info-set guestinfo.ip " + computerIP + "\"";

            proc.WindowStyle = System.Diagnostics.ProcessWindowStyle.Hidden;
            proc.UseShellExecute = false;

            System.Diagnostics.Process GetIP;
            GetIP = System.Diagnostics.Process.Start(proc);
            GetIP.WaitForExit(3000);
        }
    }
}
```

Now, as promised here's the VB equivalent of the preceding code. Simply create a new VB Console application and inset the code that follows. Remember — you do not need to do both of these. I only provide both sets of code in case you prefer one over the other. Both projects are available on the book's web site at `www.wrox.com` for download.

Program.vb

```
Module Module1

    Sub Main()
        Dim computerName As String = System.Net.Dns.GetHostName
        Dim computerIP As String =
         System.Net.Dns.GetHostEntry(computerName).AddressList(0).ToString

        setVMwareGuestInfo(computerIP)
    End Sub

    Private Sub setVmwareGuestInfo(ByVal computerIP As String)
        Dim proc As New System.Diagnostics.ProcessStartInfo("C:\Program
         Files\VMware\VMware Tools\VMwareService.exe")

        proc.Arguments = "-cmd " & Chr(34) & "info-set guestinfo.ip " & computerIP
         & Chr(34)

        proc.WindowStyle = ProcessWindowStyle.Hidden
        proc.UseShellExecute = False

        Dim GetIP As System.Diagnostics.Process
        GetIP = System.Diagnostics.Process.Start(proc)
        GetIP.WaitForExit(3000)
    End Sub
End Module
```

The flow here is simple. Use the .NET Framework methods `GetHostName()` and `GetHostEntry()` to obtain first the name and then the IP address of the guest machine. Then, make an external call to the `VMwareService.exe` file, like you did in the VBScript example, and set the IP address variable.

Now, before this is any good to you, it needs to be compiled. Select the Build menu and choose Configuration Manager. Set the active solution configuration to Release and click Close. Then, from the Build menu select Build Solution. Once that is complete, you will have a new executable named `vmGet GuestIP.exe` in the project directory. This is located under your My Documents folder in `Visual Studio 2005/Projects/<project name>/<project name>/bin/Release`. Copy the `vmGet GuestIP.exe` file from that directory into the `C:\Program Files\VMware\VMware Tools` directory.

How simple was that? In order to use this application as a VMware Tools script, repeat the same steps that you did with the VBScript example. Figures 11-1 through 11-4 should give you an easy reference.

The nice thing about the .NET solution is no fuss. It's an executable that VMware Tools can call directly, which is nice. The bad thing, for some at least, is that you have to install the .NET Framework v2.0 to use it. If that's a downside for you then use the VBScript example.

Go ahead and leave this script in place on your test machine. The integration code that follows relies on it to be in place.

CruiseControl.NET

As I alluded to before, CruiseControl.NET is a continuous integration package that effectively monitors source control for change, and when detected, it initiates a build and verification. The process itself is rather simple; however, the configuration of CCNet itself can be rather tedious. Unfortunately this chapter cannot be a guide into the proper installation and configuration of CCNet. If you perform a standard installation of CCNet, I will outline the changes I make to the default install in order for this example to function. The following are the highlights of what I want to accomplish in this section:

- Configure CCNet to retrieve the sample .NET project from Visual SourceSafe
- Build the sample .NET project using MSBuild
- If the build is successful, fire up a virtual machine and push the build onto it

This may be a little ambitious; however, it is possible. The goal of this chapter is to show you that it can be done. The next chapter outlines a real-world example that will seem a little more practical than this one.

Basic Configuration

Right off I am going to assume something. I assume that you have downloaded and installed CruiseControl.NET. You can get CCNet and view the installation instructions at `http://ccnet.thoughtworks.com`. I also assume that you have a working installation of Visual SourceSafe somewhere on your network. For the purpose of this example, it is located on the VMware Server itself.

Although I'm not covering the installation of CCNet, or Visual SourceSafe for that matter, their setup is basic and straightforward. The CCNet installation can be performed with the default install options. Nothing special is required for it to be operational for this example. The one exception is that you have ISS installed on the server prior to installing CCNet. This is to support its web interface, which I cover a little later in this chapter. As for Visual SourceSafe, its installation is very simple. Refer to the VSS documentation for how to configure a new source control database.

Adding a Development Project to Source Control

Obviously source control without any source code is rather useless so you need to add some. Of course you may have your own project to test with this setup, but in case you don't I want to walk through creating something in Visual Studio to use for the CCNet integration.

I'm going to use Visual Studio 2005 for this example, but it can easily be done with any other development tool so long as you can commit it to VSS. Open Visual Studio and either via File➪New or any other method you are provided create a new project. It doesn't matter what you create or in what language. The only thing that is important is that it compiles successfully. For this example I just created a new C# WinForms application with a single button on it. Once you are finished make sure it builds. Next comes adding it to source control.

> *The following assumes you have Visual Source Safe set up as your default source control provider. If you do not or wish to verify it, the settings can be found under Tools➪Options in the Source Control node. You may have to reveal all settings under Options to see the Source Control node. If Visual SourceSafe is not available as a source control provider, refer to the Visual SourceSafe documentation and install the VSS Client application from the installation media.*

1. As long as you have a source control provider selected under Options, the File menu should have a Source Control submenu. Click File➪Source Control to reveal the menu shown in Figure 11-5. This is how you're going to add the project to the Team Project.

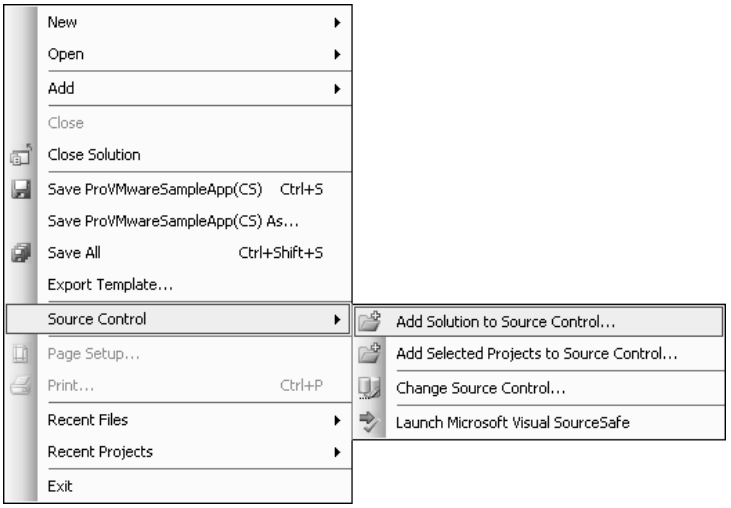

Figure 11-5

2. Next you are asked to log in to the Visual SourceSafe server, as shown in Figure 11-6. Here you provide your credentials and the database you want to connect to. If you do not have a database configured, refer to the VSS documentation to set one up and then return to this step.

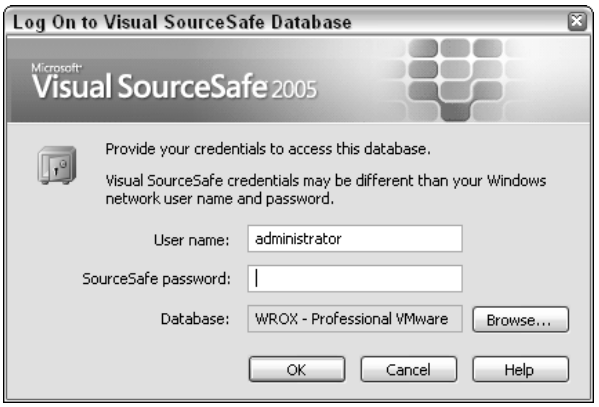

Figure 11-6

3. Select the Add Project to Source Control item to give you the dialog box shown in Figure 11-7. Because this is a new database, chances are there are no projects already defined. Name the project and click OK to create the project in source control.

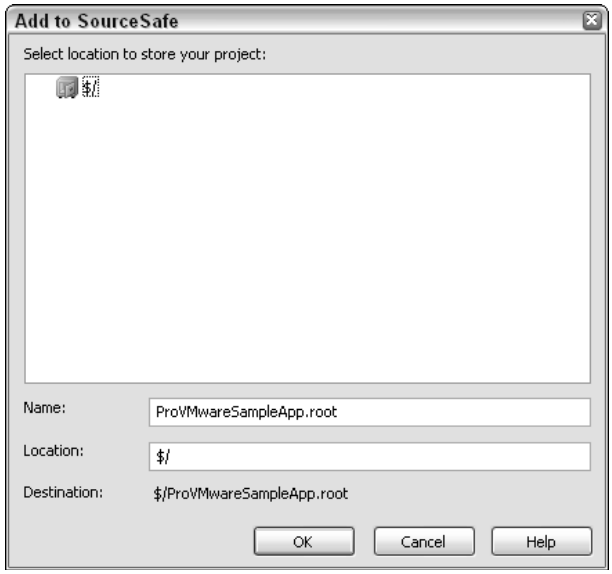

Figure 11-7

4. Depending upon the state of your project, the files may or may not have been checked into source control. The easiest way to ensure your project is under source control and everything is checked in is to check the Pending Checkins page. This is a tab along the bottom of your IDE. Click this tab to reveal the pane shown in Figure 11-8.

If the Pending Checkins tab is not visible along the bottom of your IDE, select View➪Pending Checkins➪to reveal the pane in your IDE.

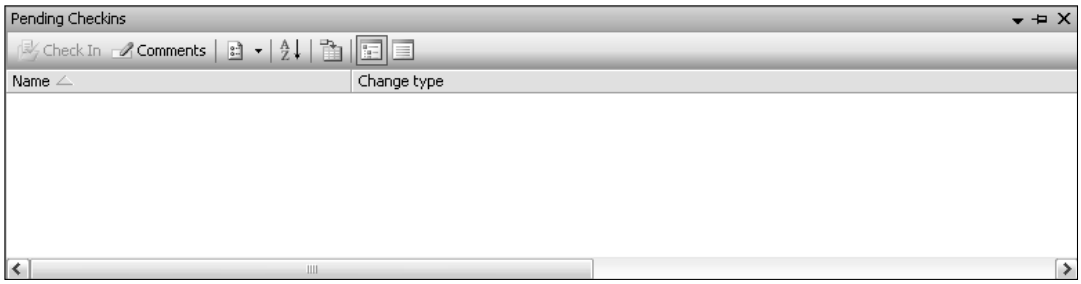

Figure 11-8

5. The files listed in the lower half of this pane are waiting to be checked into source control. Click the Check In button in the upper left of this pane when you are ready to check the project in.

You now have a working development project under source control. In the coming section you will be coming back to this project and using it during your CCNet integration.

Once you have CCNet installed, it needs to know what to do. All of the magic happens in the `ccnet.config` file located in the `C:\Program Files\CruiseControl.NET\server` directory. The default version of this file is shown in the following code block.

CCNet provides a convenient link to open this file for editing under the CruiseControl.NET folder in the Start menu.

```
<cruisecontrol>
  <!-- This is your CruiseControl.NET Server Configuration file. Add your projects
    below! -->

  <!--
    <project name="MyFirstProject" />
  -->

</cruisecontrol>
```

As you can tell this is simply an XML file where you list the projects and specific settings per project that you wish CCNet to handle. At the moment CCNet has no idea what to do, so you need to give it a little help. The project description that goes into the `ccnet.config` file falls into four sections, or categories:

- Triggers
- Source Control
- Build tasks
- Publisher tasks

Because this is XML everything has to fall within a specific set of tags, or nodes. In this case the four sections just listed all fall within the parent node *Project*. So, you can start out by defining just the project. Open the `ccnet.config` file and enter the following:

```
<cruisecontrol>

  <project name="ProVMwareSample">

  </project>

</cruisecontrol>
```

This sets the stage to configure your sample project named ProVMwareSample. Of course this by itself doesn't do a whole lot of good. Interestingly enough, however, it is all CCNet needs in order to run; it just wouldn't do anything spectacular.

In addition to the project name this node can contains a number of other project-specific settings; however, the only one that is required is the name. For a complete list of project configuration block values feel free to take a look at the CCNet web site shown previously. For the purpose of this example the defaults provided work fine.

Adding a Trigger

So you can turn to your first section, triggers. A trigger controls when CCNet will start a new cycle. The default is every 60 seconds. That means every 60 seconds CCNet will execute the tasks located inside

the <project> blocks. There are some exceptions to just what it does every 60 seconds so just be aware of this default time frame. Because that frequency is a bit extreme for this example, throttle that down a bit. You can set up this project to build at 11:00 p.m. every day. You can do this using what's called a Schedule trigger:

```
<cruisecontrol>

  <project name="ProVMwareSample">

    <triggers>

      <scheduleTrigger time="23:00" BuildCondition="ForceBuild"></scheduleTrigger>

    </triggers>

  </project>

</cruisecontrol>
```

The preceding Schedule trigger simply states that at 23:00, or 11:00 p.m., it will force a build of the project. The `BuildCondition` value can be modified to build only if a change is detected as well, but forcing a clean build is what you want so the default here is fine. If you do not specify a `BuildCondition`, the default of `IfModificationExists` is used, and the project will only build if it has changed since the last integration.

You can get the full list of the other types of triggers available from the help files for CCNet. Two of the more notable triggers you may use are the Interval and the Filter triggers:

- The **Interval trigger** is what is used by default if no other trigger is specified for the project. It starts an integration cycle after a set number of seconds has elapsed.
- The **Filter trigger** is a bit unique from the rest because it actually keeps integration from happening at or during a specific time, like during the backup of your source control system.

Adding Source Control

Okay, so now you have a project and the interval in which you want CCNet to run, but you need the actual code. This is accomplished with the Source Control block. This block is not only where you tell CCNet what project to download and build, but also where to download from. A very large number of source control systems are built in and supported by CCNet, too numerous to list here, but if you are curious feel free to check out the list on its web site at `http://ccnet.thoughtworks.com`. Because this example is using Visual SourceSafe, and it is supported, you can add some configuration values to the project definition that you've been building.

```
<cruisecontrol>

  <project name="ProVMwareSample">

    <triggers>

      <scheduleTrigger time="23:00" BuildCondition="ForceBuild"></scheduleTrigger>

    </triggers>
```

```
      <sourcecontrol type="vss" autoGetSource="true" applyLabel="true">

        <executable>"C:\Program Files\Microsoft Visual ↩
         SourceSafe\SS.EXE"</executable>
        <project>$/ProVMwareSample.root</project>
        <username>administrator</username>
        <password>pass@word!</password>
        <ssdir>C:\VSS\</ssdir>
        <workingDirectory>C:\myBuild</workingDirectory>
        <cleanCopy>true</cleanCopy>

      </sourcecontrol>

    </project>

  </cruisecontrol>
```

Because of line space restrictions in this book I was unable to show the `<executable>` *attribute correctly under the source control node. Be absolutely sure that the path present for the executable is all on a single line. Breaking it between two lines like the preceding code block will not work.*

I should probably break down some of the configuration items here. At the very least CCNet needs an executable path, project name, user name, and password for the source control server. There is, however, a catch. Unless you have the environment variable `SSDIR` defined, you also need the `<ssdir>` node, so add that to the list. Another setting I suggest is `<workingDirectory>`. This tells CCNet where to put the checked out files. I like to specify a path a little closer to root as opposed to the default, which is buried within `C:\Program Files\CruiseControl.NET\server`. Technically it provides no advantage other than less typing. So, to make things a little easier the table that follows defines what each node from the preceding code does.

Node	Description
`<executable>`	The path to the Visual SourceSafe command-line tool.
`<project>`	The name of the SourceSafe project to retrieve from the database.
`<username>`	The SourceSafe user name to connect with. This user must be able to add labels.
`<password>`	The password for the user just mentioned.
`<ssdir>`	This is the path, local or UNC, to the SourceSafe database.
`<workingDirectory>`	This is a local path to where you want the checked out files to live.
`<cleanCopy>`	This switch tells CCNet to checkout a fresh copy of the source on every run.

This is not an all-inclusive list of the options available for the Visual SourceSafe source control node. For a complete list you can reference the CCNet documentation available at `http://ccnet.thoughtworks.com`.

At this point you can actually fire up CCNet and watch it work. You need to have a project available in SourceSafe under the name listed in the `<project>` node. As long as that condition is met, CCNet should

check out the source from SourceSafe and place it into the directory you gave for the `<workingDirectory>` node. One caveat is that I told CCNet to perform an integration only at 11:00 p.m. every night. Unless your timing is perfect you will need to kick off a cycle manually. This can be accomplished with the CCNet Tray application installed along with the main CruiseControl.NET application.

Configuring the CCNet Tray Application

Because the CCNet Tray application might be a bit of a gray area for some of you, I want to take a few moments to configure it to connect to your local CruiseControl.NET server, display the project, and kick off an integration. At this point just the project files will be checked out; no build will take place. You will add a build block shortly. For now I want to concentrate on making sure you are good to go up to this point:

1. To start off you need to have the CCNet Tray application installed. This was most likely installed along with the server package. If for some reason it was not, you can install it manually. Browse to the `C:\Program Files\CruiseControl.NET\webdashboard\cctray` directory and install it via the executable present there. Once the Tray application is installed, you can move on.

The CCNet Tray application can also be installed via the web dashboard, which I cover later in this chapter.

2. The CCNet Tray application install should set it to run automatically from your startup group; if not kick it off from the Start menu now. You should get a new little CCNet icon in your task bar as shown in Figure 11-9.

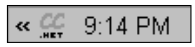

Figure 11-9

3. Normally this application would run on either your developers' machines or perhaps a system integrator or build manager. Regardless I have it running on the CCNet server itself. The functionality remains the same no matter where you put it. Go ahead and double-click the icon to reveal the application's main screen, shown in Figure 11-10.

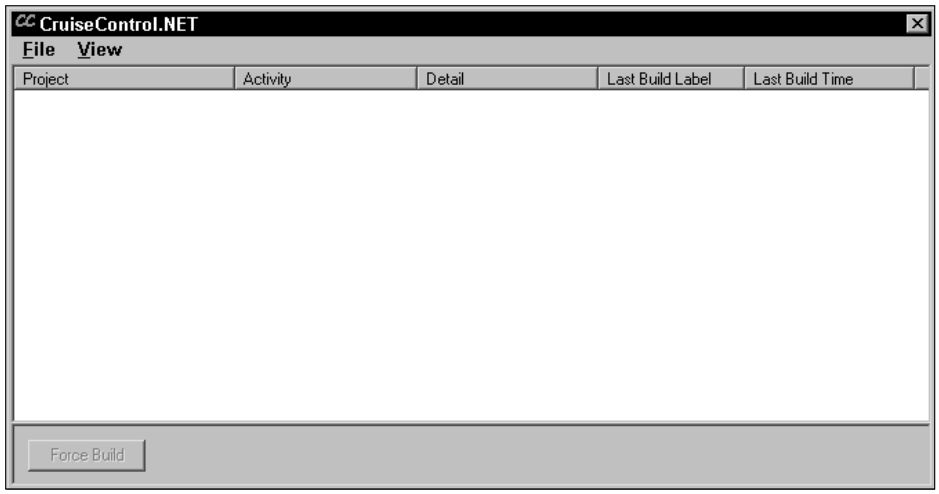

Figure 11-10

4. The sole purpose of this application is to monitor the build status of the projects you have set up on your CCNet server. Because this screen is obviously blank, you need to set up the server and projects you wish to monitor. Click the File menu followed by Settings. You should see a screen similar to the one in Figure 11-11.

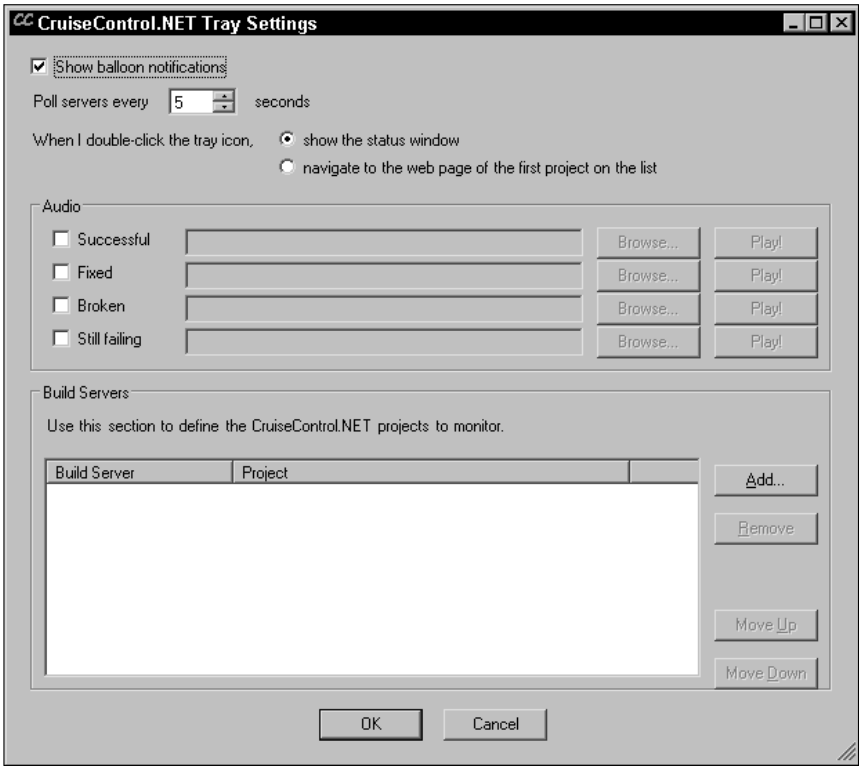

Figure 11-11

5. I'll leave most of the settings here for you to explore. For the most part they all deal with aesthetics anyway. The section you are interested in lies at the bottom of this screen, the Build Servers section. Click the Add button to the right to bring up the screen where you set your CCNet server and select your projects, as shown in Figure 11-12.

6. Because I'm using old trusty `vmserver-01` as my CCNet machine, I'll enter that into the bottom-most text box and click Add Server.

Your CruiseControl.NET server must be running during this step or it will fail. Also, if you have no configured projects there will be no project to select and therefore you will be unable to configure the Tray application at this time. You can start up the CCNet application by double-clicking the icon on your desktop or the one available in the Start menu. You can consult the CCNet online documentation at `http://ccnet.thoughtworks.com` *if you want to run it as a service.*

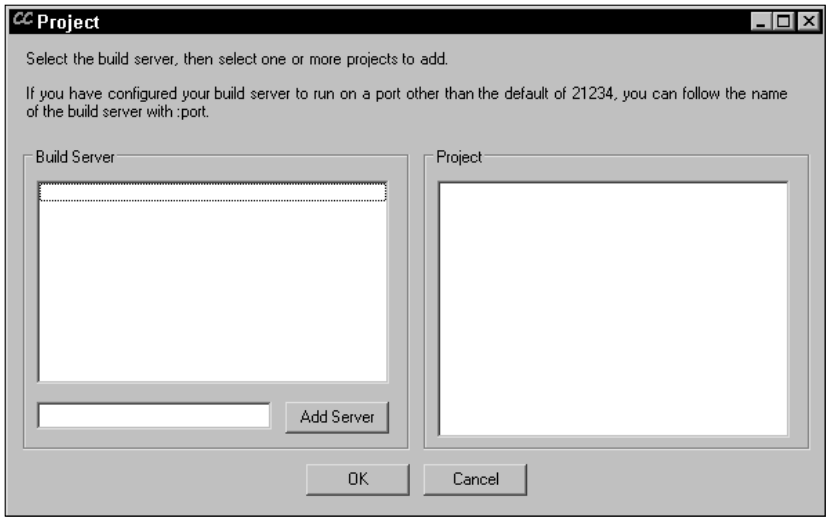
Project
Select the build server, then select one or more projects to add.
If you have configured your build server to run on a port other than the default of 21234, you can follow the name of the build server with :port.
Build Server
Project
Add Server
OK
Cancel

Figure 11-12

7. Once you enter the name of your server and click Add Server, the left-most box will contain the name of your server and the right-most box will list the projects available on that server to select from. Figure 11-13 gives an example.

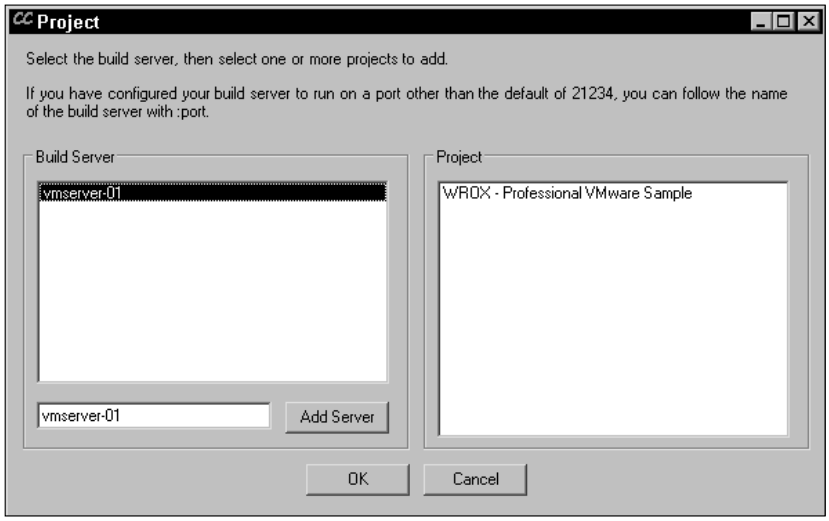
Project
Select the build server, then select one or more projects to add.
If you have configured your build server to run on a port other than the default of 21234, you can follow the name of the build server with :port.
Build Server
vmserver-01
Project
WROX - Professional VMware Sample
vmserver-01
Add Server
OK
Cancel

Figure 11-13

8. Perfect, the CCNet server is running, and the configuration file I used earlier appears to be working, at least for the moment. From here click to select the project from the right-hand pane and click OK. You will revert back to the previous screen although now there should be a server and project listed, as shown in Figure 11-14.

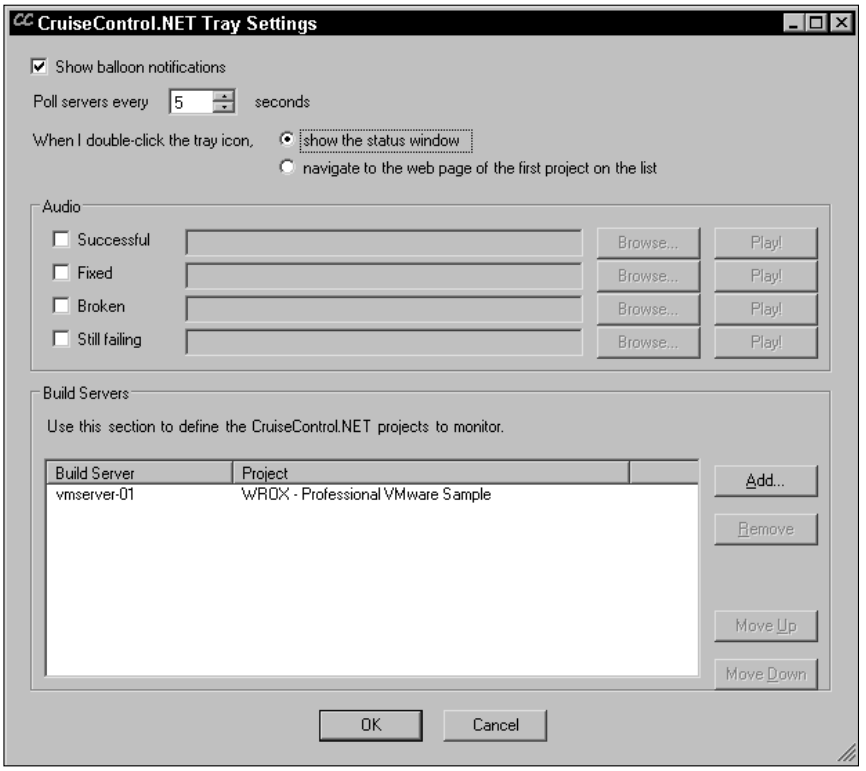

Figure 11-14

9. Go ahead and click OK on the settings screen to take you back out into the main application screen. You should now see the project you selected as well as its current build status listed. Figure 11-15 shows an example of what you might see.

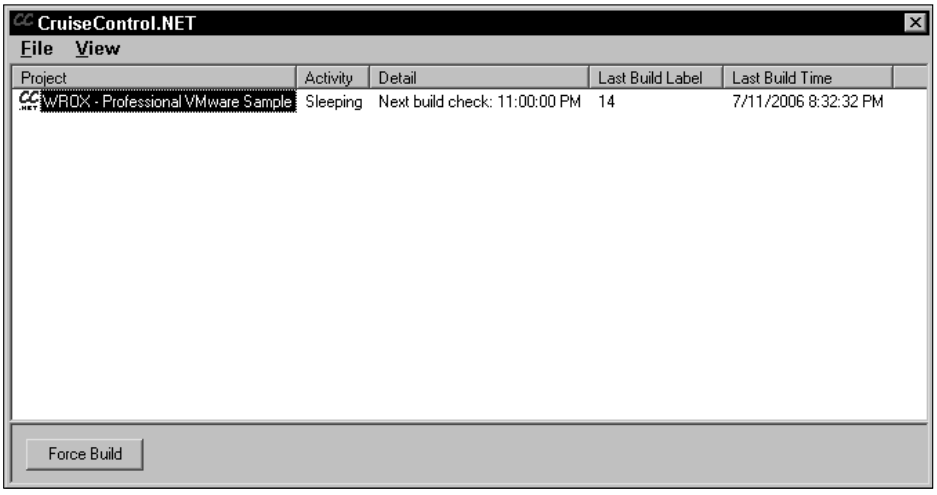

Figure 11-15

Because this is your first integration run, you will not have some of the information you see in Figure 11-15. I've been exercising the server while writing, so as you can tell from the figure, I already have some builds under my belt.

10. Click to select the project from the center pane and click the Force Build button.

If all goes well and your configuration file is correct you should see green icons and a nice bubble notification that the build was successful. You can also now check the path you used under `<workingDirectory>` to verify that indeed the project files were checked out and placed there.

Introducing the CCNet Dashboard

Now, before I move on, there's one other thing I want to take a moment to introduce you to and that's the CCNet Dashboard. If you right-click the project within the Tray application and choose Display Web Page, the web dashboard for CCNet loads displaying the project and specific information like build logs, failure logs, unit testing results, and so on. I won't actually be drilling down into the logs at this time, so just open it up and take a look around. It will be lacking a great deal of information at the moment because you have yet to actually perform a build. Figure 11-16 gives you an example of the web dashboard for the test project.

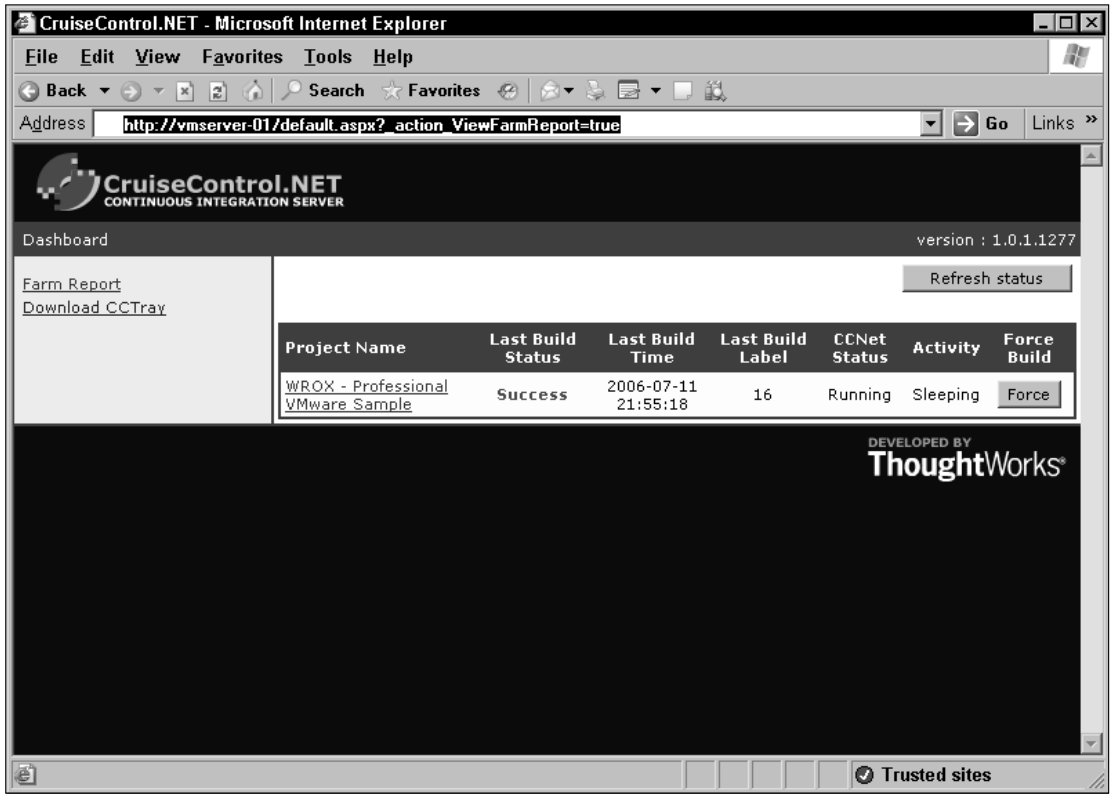

Figure 11-16

The reason I want you to see the dashboard even though you aren't using it at this moment is to give you an idea of how CCNet is set up for team use. Although the CCNet Tray application provides a quick and easy way to see build status, it lacks in reporting and detailed information. This is where the web dashboard comes in. As you will see later in the chapter once you start cranking out builds, the dashboard is your one-stop shop for all the reports and build logs generated during an integration.

Okay, so if everything up to this point has gone well, you should have a successful run under your belt — congratulations. If something did go wrong look over your `ccnet.config` file as well as your SourceSafe database to ensure the setup is correct and try again. From the web dashboard you can drill down into the project to display any exceptions or errors that occurred during the run if necessary.

Configuring Your Build

Because you have your source being checked out, it's time to do something with it. If you remember the setup for this example, you will be building the application using MSBuild, part of the .NET Framework v2.0. The MSBuild executable can be found within the framework directory typically located in `C:\Windows\Microsoft.NET\Framework\v2.0.<build number>\MSBuild.exe`. You are not limited to MSBuild, however. There are several different build methods to choose from. As before refer to the CCNet documentation on its web site for more information (`http://ccnet.thoughtworks.com`).

CCNet refers to the build and publisher block under a parent node called tasks. You will see here in a moment that hierarchy and how it relates. For now just take care that you nest build and publisher blocks within a tasks node.

True to form so far, I want to show you what you need to add to your `ccnet.config` file; then I will proceed with explaining the individual pieces. Add the following to your project configuration file.

I've included the entire file up to this point. The section to add is between the `<tasks>` nodes; however, I felt it important to display the entire configuration file in order to reduce confusion.

```
<cruisecontrol>

  <project name="ProVMwareSample">

    <triggers>

      <scheduleTrigger time="23:00" BuildCondition="ForceBuild"></scheduleTrigger>

    </triggers>

    <sourcecontrol type="vss" autoGetSource="true" applyLabel="true">

      <executable>"C:\Program Files\Microsoft Visual ↩
       SourceSafe\SS.EXE"</executable>
      <project>$/ProVMwareSample.root</project>
      <username>administrator</username>
      <password>pass@word!</password>
      <ssdir>C:\VSS\</ssdir>
      <workingDirectory>C:\myBuild</workingDirectory>
      <cleanCopy>true</cleanCopy>

    </sourcecontrol>
```

```
      <tasks>

        <msbuild>

          <executable>C:\WINDOWS\Microsoft.NET\Framework\v2.0.50727\MSBuild.exe ↩
           </executable>
          <projectFile>C:\myBuild\ProVMwareSample\ProVMwareSample.sln</projectFile>
          <buildArgs>/noconsolelogger /p:Configuration=Debug /v:diag</buildArgs>
          <timeout>15</timeout>
          <logger>ThoughtWorks.CruiseControl.MsBuild.XmlLogger,↩
           ThoughtWorks.CruiseControl.MsBuild.dll</logger>

        </msbuild>

      </tasks>

    </project>

</cruisecontrol>
```

Again due to line space limitations in this book I'm not able to properly show you on single lines the changes in the preceding code. The only item to note is the placement of the closing `</executable>` *tag. This tag must close on the same line that it was opened. The same is true for the* `</logger>` *tag. It must close on the same line in which it was opened.*

At the time of writing the most current version of the .NET Framework was v2.0.50727. You may need to adjust this to the version you have installed. Verify the version you have installed by browsing to the `C:\Windows\Microsoft.NET\Framework` *directory and looking for the v2.xxx directory.*

Probably the best way to go through this block is with a table. The following table is a list of the configuration items and their descriptions. I go into a bit more detail concerning one of them, `<logger>`, after the table.

Node	Description
`<executable>`	This is the path to the MSBuild executable. The default value for this field will most likely not be correct because versions of .NET come out quite frequently. You will want to make sure and provide this path to ensure functionality.
`<projectFile>`	This is the location of the Visual Studio solution file for the project you wish to build. Notice it references the file from within the `<workingDirectory>` path provided in the source control block.
`<buildArgs>`	This is where you can provide command-line arguments to MSBuild. At the very least you should consider the `/noconsolelogger` switch to avoid getting XML data spilled out into the console. More about capturing that XML data later in the chapter.
`<timeout>`	The amount of time in seconds to wait before assuming the process is hung. The default is 600 (10 minutes), which is a bit long.
`<logger>`	This is where you can define a custom logger assembly by class and assembly name. Most build blocks do not require this; however, MSBuild requires a separate XML logger in order to integrate the output into the web dashboard.

All of these settings are pretty straightforward except the logger. MSBuild exports its error data in an XML format that CCNet cannot consume right out of the box. I cover how to handle that in the next section.

Adding Support for MSBuild XML Logging

In order to incorporate MSBuild data into your web dashboard a few simple steps are required.

> *These steps are subject to change dependent upon the version of CCNet you use and its development path. Be sure and check the online documentation for possible updates to this procedure. The version of CCNet used in this example and to provide these specific steps is v1.0.1.1277.*

1. Download the MSBuild XML logger assembly from the CCNet web site (`http://ccnet.thoughtworks.com`). Search for Using CruiseControl.NET with MSBuild to get the correct location of the file to download.
2. Extract and copy the .DLL file into your working directory for the project. This, however, is where the term *working directory* gets a bit clouded. Up to this point I have referred to the working directory in terms of source control, which is `C:\myBuild` in this example. The working directory you need here is actually the project's working directory. Because you did not specify a custom working directory for the project, you can find it in its default location under `C:\Program Files\CruiseControl.NET\server\<project name>`. Go into the `WorkingDirectory` beneath. This is where you want to drop the `ThoughtWorks.CruiseControl.MsBuild.dll` file you downloaded. Any other location and the build will fail.

Now, you have the custom logger for MSBuild downloaded and placed in its proper location, and unfortunately, this is something you will have to do manually at the moment for each and every project that utilizes MSBuild.

Next, you need to configure the web dashboard to display MSBuild log files for your project:

1. Under the `C:\Program Files\CruiseControl.NET\webdashboard` directory open the `dashboard.config` file.
2. Within the `<buildReportBuildPlugin>` node locate the `<xslFileNames>` node. Within that node add the following:

```
<xslFile>xsl\compile-msbuild.xsl</xslFile>
```

3. Finally, add the following with the rest of the `<xslReportBuildPlugin>` entries:

```
<xslReportBuildPlugin description="MSBuild Output"
actionName="MSBuildOutputBuildPlugin" xslFileName="xsl\msbuild.xsl" />
```

Once that is complete you will now capture and display MSBuild output within the web dashboard.

You are finished with the `dashboard.config` file. Save and close it now. The remaining steps all take place back in the `ccnet.config` file where your project configuration lives.

Adding a Build Distribution Node

One final thing to add to your configuration file at this time is a build distribution node. This will copy the contents of the build into a directory of your choosing on the server. The purpose of this will become

apparent later when you distribute that build up to the VMware Server virtual machine. Add the following to your configuration file after the MSBuild section:

```
        <buildpublisher>
          <sourceDir>C:\myBuild\ProVMwareSample\ProVMwareSample\bin\Debug</sourceDir>
          <publishDir>C:\</publishDir>
        </buildpublisher>
```

This performs little more than a simple copy operation. It takes the contents of the `C:\myBuild\ProVMwareSample\ProVMwareSample\bin\Debug` directory and copies it into a directory under `C:\` named after the build label CCNet automatically applied to this run. You will use this copy later in the script to move the project's binaries up to the virtual machine.

So this would be a good time to test your configuration and see how everything turns out. Open the CCWeb Tray application and force a build. If everything goes well you should get that friendly bubble letting you know that yet another successful build took place. If not, check your configuration and drill down into the web dashboard for some clues as to why the build failed.

Because the basic configuration is down and hopefully providing successful builds, it's time to move into integrating with VMware Server.

VMware Server Integration

There comes a point in every development cycle where your quality team, or whoever happens to be filling that role, needs to start taking a look at the iterations flowing out from your development team. This addition of quality review to your development process can get a bit messy especially if it is done by a group outside the core developer circle. A couple of the more common problems I have personally witnessed in the past are stale code and lack of coordination. You can just nip that in the bud by pushing a build out for them onto a fresh test bank VM. (Of course they may just ask where their coffee is if you do too much for them.)

So now it is time to flex your VMware Server API skills to their extreme. Through the use of something CCNet calls an Executable Task and your Perl scripts, you can accomplish a great deal. The following quickly outlines the path you are going to take:

1. Insert an `<exec executable>` task into your CCNet project block to launch your external Perl script.
2. Capture the `CCNetWorkingDirectory` environment variable exposed by CCNet to locate your project.
3. Locate a virtual machine that's part of your test bank that is not already in use and power it on.
4. Grab the virtual machine's IP address through the use of the `$vm->get_guest_info` method on the host and a client-side script.
5. Push the build onto the virtual machine's desktop.
6. Notify the user, or group of users, with the name of the virtual machine and status of the package that's ready to test.

That's a tall order. This can, however, be done, and you'll look great when you pull it off.

There will be two elements to this, the Perl script and the CCNet project hook. The following is a list of what you want to accomplish with your Perl script and your VMware Server:

- Connect to your VMware Server and retrieve a list of registered virtual machines.
- Parse that list for virtual machines that match your project's needs (for example Testing images, Windows XP, and so on).
- Connect to each matching virtual machine and determine its status.
- Power on the first machine available.
- Through the use of VMware Tools scripting and `$vm->get_guest_info` obtain the IP address of the virtual machine in use.
- Copy the project payload up to the virtual machine's desktop.
- Notify the QA team of the virtual machine's name and status.

In the past I've just discussed the code sample and then provided the entire piece of code. Here I will change things up a bit because I want to discuss what's going on over a few iterations of the script. I will, however, provide the entire script at the end of this section so you can see everything together.

Retrieving a List of Registered Virtual Machines

So first, the easy stuff, connecting to the VMware Server and retrieving a list of virtual machines currently registered. The script assumes that your testing images are already located in inventory, which is normally the case, and that you have followed a standard naming scheme for each. If you remember from Chapter 6 I told you the naming would come into play later. This is where standard naming schemes for your machines becomes vital.

This first section will connect to your server and retrieve a list of virtual machines into an array. There will be no passing of connection arguments to this script although you could if you really wanted to. Refer to the examples in Chapter 9 if you want to pass them in via the command line. For this example I will simply hard code them.

The VMware Server being used here is on the same machine as CruiseControl.NET.

```
#!/usr/bin/perl -w

# This is where you add the Windows specific libraries if the platform is MSWin32
BEGIN {
  if ($^O eq "MSWin32") {
     @INC = (
        'C:\Program Files\VMware\VMware VmPerl Scripting
          API\perl5\site_perl\5.005',
        'C:\Program Files\VMware\VMware VmPerl Scripting
         API\perl5\site_perl\5.005\MSWin32-x86');
  }
}

use strict;

# Import the VMware Perl modules needed
```

```
use VMware::VmPerl;
use VMware::VmPerl::Server;
use VMware::VmPerl::VM;
use VMware::VmPerl::ConnectParams;

# This extracts the integer value for constants out of the API
# Use this as a quick helper method to perform integer to constant string
# conversions
# This is adapted from one of the VMware samples and I use it in almost every
# script.
sub constant_to_string {
  my $constant_string = shift;
  return VMware::VmPerl::constant($constant_string, 0);
}

my $state_string_map = {};
my @state_strings = ("VM_EXECUTION_STATE_ON",
                     "VM_EXECUTION_STATE_OFF",
                     "VM_EXECUTION_STATE_SUSPENDED",
                     "VM_EXECUTION_STATE_STUCK",
                     "VM_EXECUTION_STATE_UNKNOWN");

# Loop through the strings above and pull out the integer value for each from the
# API
foreach my $state_string (@state_strings) {
  $state_string_map->{constant_to_string($state_string)} = $state_string;
}
```

The preceding is just all the preliminary stuff. If you remember from the Chapter 9 examples, this helps set up your environment, parse the API constants, and convert them to strings for use later. You might also recall that this will come in very handy in a bit when you start checking the status of your machines. Honestly this block should be part of every Perl script you write against the VMware APIs.

Now you can move on, connect to the server, and grab some machine names:

```
my $server_name = "localhost";
my $port = 902;
my $username = "administrator";
my $password = "Pass@word1";

# Create our $connectparams object to use when we connect to the server
my $connect_params =
VMware::VmPerl::ConnectParams::new($server_name,$port,$username,$password);

# Create the server object
my $server = VMware::VmPerl::Server::new();

# Connect to the server or display the error on failure
if (!$server->connect($connect_params)) {
  my ($error_number, $error_string) = $server->get_last_error();
  die "Count not connect to server: Error $error_number: $error_string\n";
}

# Grab a list of the available virtual machines in inventory and place them into an
```

```
# array
my @vm_list = $server->registered_vm_names();

# Make sure we actually have a list to use.
if (!defined($vm_list[0])) {
  my ($error_number, $error_string) = $server->get_last_error();
  die "Could not get a list of virtual machines from the server: Error
       $error_number
      : $error_string\n";
}
```

Right off you might ask, why not just call `ConnectParams::new()` without any arguments? Technically that would work as long as the user under which CCNet is executing the Perl script has administrative permission to access the VMware Server. Because this might not be the case it is safer to just pass the information into the `new()` method even if the script is running local, on a default port, and is running under a context that has the proper access to the VMware Server. This offers a little flexibility if in fact the preceding stipulations are not met.

Next, I want talk about error handling. As you can see in the code, you have error handling for the `$server->connect()` method and to make sure you actually have at least one virtual machine available on the server. Any failure with your Perl script and CCNet will fail the entire build. That's actually a good thing because you wouldn't want the build to succeed if the script failed. Really what CCNet is looking for is a return value on exit of `0`, which is the standard return value for Perl if everything went okay. Don't let this shy you away from actually returning information from your script. CCNet takes such output and adds it to the build log. Only an exit value of 1, an error, will trigger a failure. The use of `die()` in your error routines exits the script with a value of 1 and triggers a failure. CCNet and Perl are playing nice so far, so you can push on.

Parsing the List of Virtual Machines

The next section of code will parse the array of virtual machines registered on the server via a regular expression. For the sake of example assume you want to test this on Windows XP with SP2. This means you will filter the registered machines by `test-winxppro-sp2`.

```
# Iterate through array of virtual machines and scoop out ones that match # the
regex
my @vm_matching;
my @vm_available;

foreach (@vm_list) {
  if (/test-winxppro-sp2.*/) {
    push @vm_matching, $_;
  }
}

# Now that we have a list that match our criteria let's check their status

foreach (@vm_matching) {

  my $vm = VMware::VmPerl::VM::new();

  if (!$vm->connect($connect_params, $_)) {
    my ($error_number, $error_string) = $vm->get_last_error();
```

```
      die "Could not connect to the virtual machine: Error $error_number :
          $error_string\n";
    }

    my $power_state = $vm->get_execution_state();

    if (!defined($power_state)) {
      my ($error_number, $error_string) = $vm->get_last_error();
      die "Could not get the virtual machine's power state: Error $error_number :
          $error_string\n";
    } elsif ($power_state == VM_EXECUTION_STATE_OFF) {
      push @vm_available, $_;
    }

    undef $vm;
  }

  if (!defined(@vm_available)) {
    die "No virtual machines are available that meet the criteria for this test.\n";
  }
```

Some of you die-hard Perl programmers out there will most likely cringe at this block. I was purposely verbose with this in the interest of learning, so bear with me.

You already have a list of registered virtual machines in the `@vm_list` array. Now you need to sort them out by the requirements for this project. I've created two arrays, `@vm_matching` and `@vm_available`. The script iterates through the entire list of virtual machines on the server from the `@vm_list` array and places the ones that satisfy your regular expression into the `@vm_matching` array.

Now, I should take a moment out here to discuss what exactly the regular expression is doing for you. If you remember back in Chapter 6 I hinted pretty hard about how important a common naming scheme was for your library. The pretend design specifications for this sample call for it to be tested on Windows XP Professional with SP2. Normally of course your test platform would be a lot more diverse; however, for this example I need to at least try and keep it somewhat simple. So if you followed my suggested naming scheme earlier in the book your test images all begin with `test` followed by the operating system followed by the patch level. The pattern was `<image type>-<operating system>-<patch level>`. This is where consistency comes in. Because your spec calls for Windows XP Professional with SP2, you know you're looking for images that start with `test-winxppro-sp2`.

> *If you choose to add some naming after the patch level, be aware that you have to deal with that in your regular expression.*

Taking the regular expression from the preceding code (`/test-winxppro-sp2.*/`), I will outline in the following table just what gets accepted as a match to help you understand the process.

String (path to configuration file)	Match
`C:\VirtualMachines\test-winxppro-sp2\test-winxppro-sp2.vmx`	Yes
`C:\VirtualMachines\test-winxppro-sp2-1231\test-winxppro-sp2-1231.vmx`	Yes
`C:\VirtualMachines\base-winxppro-sp2\base-winxppro-sp2.vmx`	No

As you can see the regular expression used matches anything that at least starts with *test-winxppro-sp2*. You have to be careful here and have a firm and consistent naming scheme because you can easily grab a virtual machine that doesn't meet your specs.

Now, without getting into exactly how the rules progress with the Perl Regex (regular expressions) engine I will tell you that the expression I used here matches starting at the first `test-winxppro-sp2`, which is actually the directory name and not the configuration file name. This does not matter, however, because you used a consistent naming scheme. The script isn't saving what it matches. It's just using it to decide what to push into the `@vm_matching` array.

If you return to the script, you can see that the script iterates through the array of matching virtual machines for this project and polls them for their power state. You are only interested in used virtual machines that are in the OFF state. Machines that return an ON or SUSPENDED state are assumed to be busy and are not considered for use. If the machine is in fact in the OFF state, the script pushes it into the `@vm_available` array where you will pick it up later.

In the final section the scripts check to see if any virtual machines on the server were available for use by checking if the `@vm_available` array contains any values. Here is where you deliberately cause the build to fail if none are found. Because the goal of your CCNet script is to build and push to a server, then it stands to reason that if none are available you fail.

Powering on the First Available Machine

You're almost there. Now that you have an array that contains a list of what virtual machines are available for use that fit your testing criteria you can continue on. This next block just pops out the first one that is in the `@vm_available` array and powers it on. In a real situation you might want to control just what virtual machine is chosen here, but for the purpose of this example the first one in the array is fine.

```
# Grab an available vm from our array
my $vm_selected = pop @vm_available;

# Create a new VM object
my $vm = VMware::VmPerl::VM::new();

# Connect to the vm
if (!$vm->connect($connect_params, $vm_selected)) {
  my ($error_number, $error_string) = $vm->get_last_error();
  die "Could not connect to the virtual machine: Error $error_number :
      $error_string\n";
}

# Start the vm up
if (!$vm->start()) {
  my ($error_number, $error_string) = $vm->get_last_error();
  die "Could not power on the virtual machine: Error $error_number :
       $error_string\n";
}
```

This is a nice simple block. The script pops out an available virtual machine from the `@vm_available` array and places it into `@vm_selected`. Create a new VM object, connect, and fire it up.

Retrieving the Virtual Machine's IP Address

So far you've just scripted some pretty basic stuff, some of which you've already seen in Chapter 9. Now you are going to spice it up a little. Here you use the client-side script you created in the "VMware Tools Scripting" section earlier in the chapter to push your guest's IP address into a `guestinfo` variable. Because you took care of the client end already, you can now add what you need to the Perl script to retrieve that information and use it in your file copy.

```
# Time to put the script to sleep for about 45 seconds to give the guest time to
# startup and set the variable

print "VM Starting...\n";
sleep(45);

# Now that the guest has had time to get going let's grab the IP address from the
# guestinfo.ip variable

my $ip = $vm->get_guest_info('ip');

if (!defined($ip)) {
  my ($error_number, $error_string) = $vm->get_last_error();
  undef $vm;
  die "Could not get IP address: Error $error_number: $error_string\n";

  print "The VM's IP address is:\n$ip\n";

}
```

Right off you have to put Perl to sleep for a bit to allow time for the virtual machine to startup and run our script. About 45 seconds for a clean Windows XP Professional guest worked well with my tests. Next you pull the IP information the client script placed into the `guestinfo.ip` variable with `$vm->get_guest_info('ip')`. This places it into a scalar named `$ip`. After a little error checking you are ready to move into your final phase—pushing the build products up to the virtual machine's desktop.

Pushing the Project Payload to the Virtual Machine's Desktop

Pay close attention to where I place the next code block. This will actually be inside an `else` statement from the final `if` block from the preceding section. I'll reprint the entire `if` block to avoid confusion.

The final section of code assumes you have File and Printer sharing enabled in the guest machine's firewall. This is required for a copy in the administrative share `c$` to work.

```
if (!defined($ip)) {
  my ($error_number, $error_string) = $vm->get_last_error();
  undef $vm;
  die "Could not get IP address: Error $error_number: $error_string\n";
} else {

  # This is the command line setup for net use.  You must escape all \ and $
  # within.
  my @args = ("net use ", "x: ", "\"\\\\$ip\\c\$\\Documents and
```

```
                Settings\\Administrator\\Desktop\" ", "/user:administrator ",
                "Pass\@word!;

    system (@args);

    # This is our copy command
    @args = ("xcopy ", "C:\\$ENV{CCNETLABEL}\\*.* ", "x:\\$ENV{CCNETLABEL}\\");

    system (@args);

    # This is removing the network drive we setup for the copy
    @args = ("net use ", "x: ", "/delete");

    system (@args);
  }

  undef $vm;
  undef $server;
```

> **Make sure you add your own specific authentication information to the preceding block that corresponds to your virtual machine setup.**

Perl doesn't allow you an easy way to copy files across Windows shares with authentication involved, so you have to use a couple of external calls to accomplish it. This is done with a series of arguments placed into the `@args` array and then passed into the `system()` method. I do it this way mainly for ease because you can separate the arguments into their own elements in the array. Next you set up the copy command and use an environment variable that CCNet graciously provides for you. The `CCNetLabel` variable is the label that CCNet assigns to the build. The result of this command copies everything from the project's `bin` directory into a directory on the virtual machine's desktop named for the label of the current build.

Reviewing the Final Script

Just for review, take a look at just what you accomplished in the preceding script:

- Connecting to the VMware Server
- Grabbing a list of registered virtual machines
- Sorting the list based on the criteria set forth in the regular expression
- Powering on the first virtual machine in the sorted list that's available
- Retrieving the IP address from the VMware Tools script
- Copying the project binaries up to the virtual machine's desktop

The following lists the entire script now as a complete unit to help avoid confusion.

vmTestBank.pl

```
#!/usr/bin/perl -w

# This is where you add the Windows specific libraries if the platform is MSWin32
BEGIN {
  if ($^O eq "MSWin32") {
     @INC = (
        'C:\Program Files\VMware\VMware VmPerl Scripting
          API\perl5\site_perl\5.005',
        'C:\Program Files\VMware\VMware VmPerl Scripting \
          API\perl5\site_perl\5.005\MSWin32-x86');
  }
}

use strict;

# Import the VMware Perl modules needed
use VMware::VmPerl;
use VMware::VmPerl::Server;
use VMware::VmPerl::VM;
use VMware::VmPerl::ConnectParams;

# This extracts the integer value for constants out of the API
# Use this as a quick helper method to perform integer to constant string
# conversions
# This is adapted from one of the VMware samples and I use it in almost every
# script.
sub constant_to_string {
  my $constant_string = shift;
  return VMware::VmPerl::constant($constant_string, 0);
}

my $state_string_map = {};
my @state_strings = ("VM_EXECUTION_STATE_ON",
                     "VM_EXECUTION_STATE_OFF",
                     "VM_EXECUTION_STATE_SUSPENDED",
                     "VM_EXECUTION_STATE_STUCK",
                     "VM_EXECUTION_STATE_UNKNOWN");

# Loop through the strings above and pull out the integer value for each from the
# API
foreach my $state_string (@state_strings) {
  $state_string_map->{constant_to_string($state_string)} = $state_string;
}

######################################################################

my $server_name = "localhost";
my $port = 902;
my $username = "administrator";
my $password = "Pass\@word!";
```

(continued)

vmTestBank.pl *(continued)*

```
# Create our $connectparams objct to use when we connect to the server
my $connect_params = VMware::VmPerl::ConnectParams::new($server_name, $port,
                                                        $username, $password);

# Create the server object
my $server = VMware::VmPerl::Server::new();

# Connect to the server or display the error on failure
if (!$server->connect($connect_params)) {
  my ($error_number, $error_string) = $server->get_last_error();
  die "Count not connect to server: Error $error_number: $error_string\n";
}

# Grab a list of the available virtual machines in inventory and place them into an
# array
my @vm_list = $server->registered_vm_names();

# Make sure we actually have a list to use.
if (!defined($vm_list[0])) {
  my ($error_number, $error_string) = $server->get_last_error();
  die "Could not get a list of virtual machines from the server: Error
       $error_number : $error_string\n";
}

#########################################################################

# Iterate through array of virtual machines and scoop out ones that match the regex
my @vm_matching;
my @vm_available;

foreach (@vm_list) {
 if (/test-winxppro-sp2.*/) {
    push @vm_matching, $_;
  }
}

# Now that we have a list that match our criteria let's check their status

foreach (@vm_matching) {

  my $vm = VMware::VmPerl::VM::new();

  if (!$vm->connect($connect_params, $_)) {
  my ($error_number, $error_string) = $vm->get_last_error();
  die "Could not connect to the virtual machine: Error $error_number :
      $error_string\n";
  }

  my $power_state = $vm->get_execution_state();

  if (!defined($power_state)) {
    my ($error_number, $error_string) = $vm->get_last_error();
```

```
    die "Could not get the virtual machine's power state: Error $error_number :
        $error_string\n";
  } elsif ($power_state == VM_EXECUTION_STATE_OFF) {
    push @vm_available, $_;
  }

  undef $vm;

}

if (!defined(@vm_available)) {
    die "No virtual machines are available that meet the criteria for this
        test.\n";
}

#########################################################################

# Grab an available vm from our array
my $vm_selected = pop @vm_available;

# Create a new VM object
my $vm = VMware::VmPerl::VM::new();

# Connect to the vm
if (!$vm->connect($connect_params, $vm_selected)) {
  my ($error_number, $error_string) = $vm->get_last_error();
  die "Could not connect to the virtual machine: Error $error_number :
      $error_string\n";
}

# Start the vm up
if (!$vm->start(VM_POWEROP_MODE_SOFT)) {
    my ($error_number, $error_string) = $vm->get_last_error();
    die "Could not power on the virtual machine: Error $error_number :
         $error_string\n";
}

#########################################################################

# Time to put the script to sleep for about 45 seconds to give the guest time to
# startup and set the variable
print "VM Starting...\n";
print "Sleeping for 60 seconds.\n";
sleep(60);

# Now that the guest has had time to get going let's grab the IP address from the
# guestinfo.ip variable

my $ip = $vm->get_guest_info('ip');

if (!defined($ip)) {
  my ($error_number, $error_string) = $vm->get_last_error();
  undef $vm;
  die "Could not get IP address: Error $error_number: $error_string\n";
} else {
```

(continued)

vmTestBank.pl *(continued)*

```
    # This is the command line setup for net use.  You must escape all \ and $
    # within.
    my @args = ("net use ", "x: ", "\"\\\\$ip\\c\$\\Documents and
                 Settings\\Administrator\\Desktop\" ", "/user:administrator ",
                "Pass@word!");

    system (@args);

    # This is our copy command
    @args = ("xcopy ", "C:\\$ENV{CCNETLABEL}\\*.* ", "x:\\$ENV{CCNETLABEL}\\");

    system (@args);

    # This is removing the network drive we setup for the copy
    @args = ("net use ", "x: ", "/delete");

    system (@args);
  }

  undef $vm;
  undef $server;
```

Finishing the CCNet Configuration File

To wrap this up you have one thing left to do in your CCNet configuration file, which was left unfinished. Go back to it now. Open the `ccnet.config` file you were working on before and add the following under the MSBuild block at the end of the file:

```
        <exec>
          <executable>perl</executable>
          <buildArgs>"C:\Program Files\CruiseControl.NET\server\WROX - ➲
                     Professional VMware ➲
                     Sample\WorkingDirectory\vmTestBank.pl"</buildArgs>
        </exec>
```

Again line space for this code block was limited so be sure and place the entire attribute for `<buildArgs>` *on a single line.*

This tells CCNet to call the Perl script just after the build has completed. It should be noted that if the build fails, this block will never get executed.

The CCNet configuration file shows the path to your Perl script as the working directory for the project. Either copy your Perl script here or adjust the path in the `ccnet.config` *file to match your location.*

That final block is all that was left of the setup. The following is the entire project node you added to the `ccnet.config_file` for this example.

ccnet.config

```
<cruisecontrol>

  <project>
```

```
      <name>WROX - Professional VMware Sample</name>
      <triggers>
        <scheduleTrigger time="23:00" buildCondition="ForceBuild"></scheduleTrigger>
      </triggers>

      <sourcecontrol type="vss" autoGetSource="true" applyLabel="true">
        <executable>"C:\Program Files\Microsoft Visual
                    SourceSafe\SS.EXE"</executable>
        <project>$/ProVMwareSample.root</project>
        <username>administrator</username>
        <password>Pass@word!</password>
        <ssdir>C:\VSS\</ssdir>
        <workingDirectory>C:\myBuild</workingDirectory>
        <cleanCopy>true</cleanCopy>
      </sourcecontrol>

      <tasks>
        <msbuild>
          <executable>C:\WINDOWS\Microsoft.NET\Framework\v2.0.50727\MSBuild.exe
          </executable>
          <projectFile>C:\myBuild\ProVMwareSample\ProVMwareSample.sln</projectFile>
          <buildArgs>/noconsolelogger /p:Configuration=Debug /v:diag</buildArgs>
          <timeout>15</timeout>
          <logger>ThoughtWorks.CruiseControl.MsBuild.XmlLogger,
                  ThoughtWorks.CruiseControl.MsBuild.dll</logger>
        </msbuild>

        <buildpublisher>
          <sourceDir>C:\myBuild\ProVMwareSample\ProVMwareSample\bin\Debug</sourceDir>
          <publishDir>C:\</publishDir>
        </buildpublisher>

        <exec>
          <executable>perl</executable>
          <buildArgs>"C:\Program Files\CruiseControl.NET\server\WROX - Professional
                      VMware Sample\WorkingDirectory\vmTestBank.pl"</buildArgs>
        </exec>

      </tasks>

    </project>
  </cruisecontrol>
```

Depending upon your host server's configuration you may have to specify the full path to the Perl executable in the `<exec>` *node. It is normally located in the* `C:\Program Files\VMware\Vmware VmPerl Scripting API/` *directory. This is not normally necessary but may be required for your configuration.*

The task of basic VMware Server integration into your continuous integration environment provided by CruistControl.Net is now complete. Feel free now to kick off a build with CCNet and view the results. In the end you should have a project folder on the desktop of your virtual machine named after the source control label containing only the project's binaries. The result is a nice, neat build that is all ready for QA in the morning.

Visual Studio Team System

Visual Studio Team System, or VSTS, is a new guy on the block. Microsoft released it along with its Visual Studio 2005 and .NET Framework v2.0 line in late 2005. Though VSTS is a great product in terms of team collaboration and coordination, it is not a continuous integration product like CCNet in the preceding section. VSTS builds, or *team builds* as it refers to them, are simply build configurations set up in the team folders for the project that you can kick off manually. That being said, there are ways in which you can automate a team build; however, that functionality is not built into the product natively.

You can start by creating just a simple team project made up of one C# WinForms application. You can obtain the sample application I'm using here from the book's web site at `www.wrox.com`; however, it is little more than a simple Hello World application.

Creating a Team Project

The first thing you need to do is set up a new team project:

1. From the Visual Studio IDE open the Team Explorer from the View menu so the IDE looks like Figure 11-17.

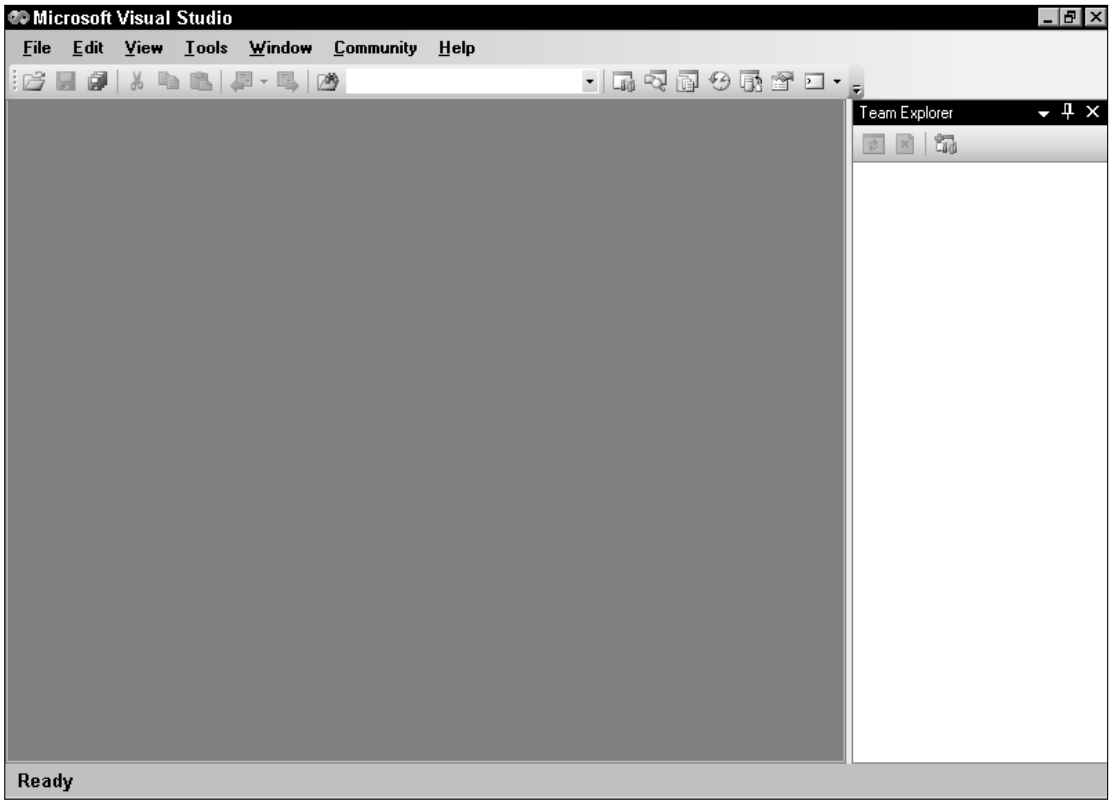

Figure 11-17

2. Now, you need to connect to the VSTS server to get things rolling. Select Tools➪Connect to Team Foundation Server, as shown in Figure 11-18.

Figure 11-18

3. This reveals the Team Foundation connection dialog. From here you can connect to several different Team Foundation Server (TFS) servers at once. What you need to do is select the Add button to add the TFS server you want to connect to. Figure 11-19 shows the dialog you are looking for.

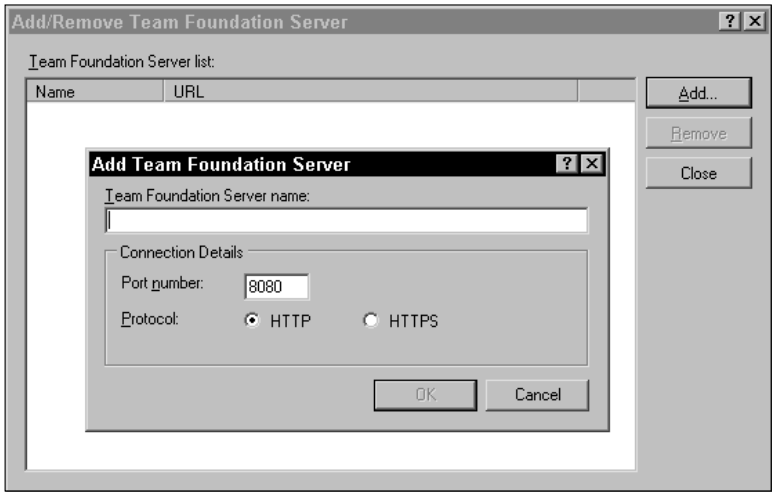

Figure 11-19

4. The default port and protocol are fine unless you know that your install uses different information. Simply enter the name of your TFS server in the box and click OK. If the connection was made, you should see something similar to Figure 11-20 with your TFS server name filled in.

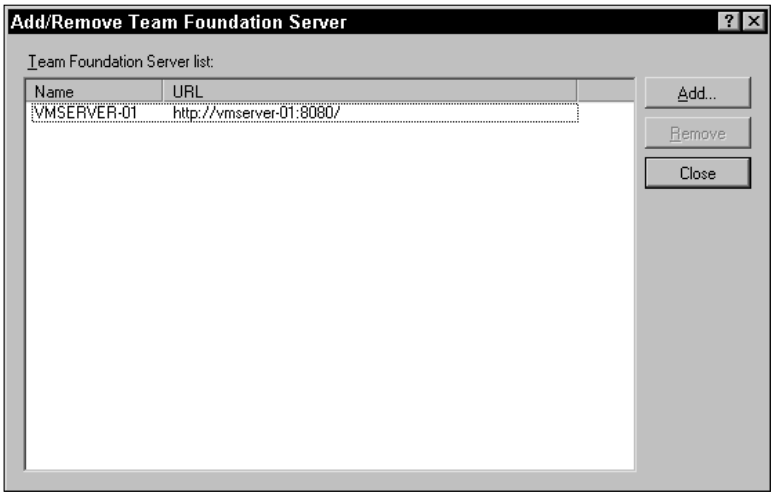

Figure 11-20

5. Because the example TFS server is brand new, there are no projects currently on it; however, yours may be different. Regardless click the Close button to get back to the parent. If there were existing projects on the server, you could select from this screen what projects you wanted to connect to before clicking OK. For this example, you don't have any existing projects, so just click OK and you'll add a new one. Figure 11-21 shows the server connection window.

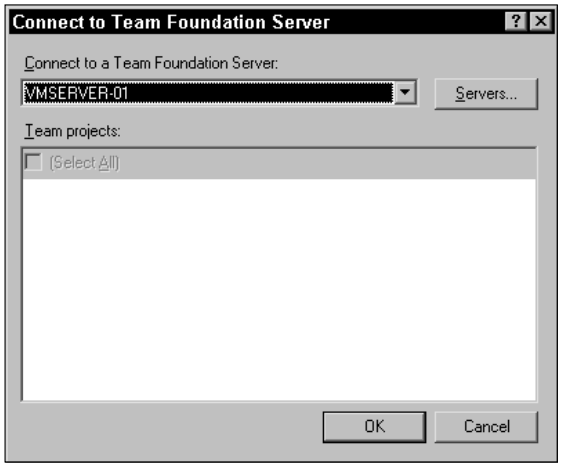

Figure 11-21

6. As long as everything is in order you should see the server displayed in the Team Explorer. This pane is where you control all aspects of your TFS project and where you spend the majority of your time. Make sure your IDE looks similar to Figure 11-22, and then right-click the server name and select New Team Project.

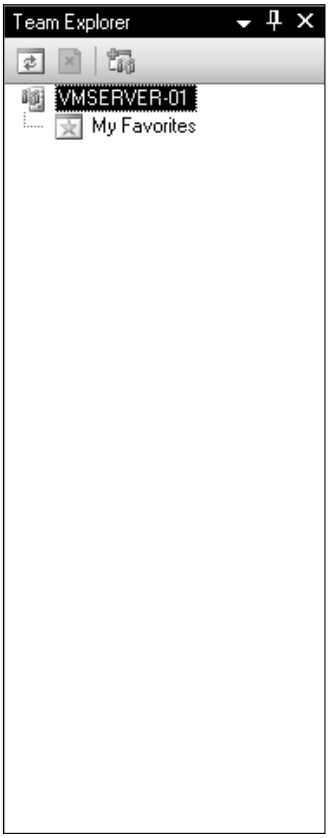

Figure 11-22

7. You should now be looking at the New Team Project wizard, shown in Figure 11-23. This is where you define the basic setup of a team project. Very little can be done from this wizard, but I'll step through the process just for the sake of example.

New Team Project on http://vmserver-01:8080
Specify the Team Project Settings
The New Team Project Wizard uses the team project name you type here when creating various components in Visual Studio Team System. After the team project is created, the name is used by team members to locate the team project in Team System. Be sure the name you pick is unique and not already in use on:
- Team Foundation Server
- Windows SharePoint Services
- SQL Server Reporting Services
What is the name of the team project?
ProVMwareSample
< Previous
Next >
Finish
Cancel

Figure 11-23

8. Start by entering a project name. For this example name the project `ProVMwareSample`. Enter your project name in the box, and click the Next button.

9. The next step of the wizard asks what type of process you want your development team to follow. I'm not even going to get into what you should select here, but there are only two choices. Agile works just fine. Select the one you want and click Next again.

10. You should now be sitting at Figure 11-24.

    In this step the wizard is asking for the name you wish to give the project portal. The project portal it is referring to is the Windows SharePoint Services site it creates for each project. Enter whatever you like here and click Next.

11. The next step asks you to configure source control for the project, as shown in Figure 11-25.

    Because you want source control for this project, the default is acceptable. Take note of the options here, however. If you continue to utilize TFS, you will most likely be at this screen frequently. Leave the top choice selected to create source control and click Next.

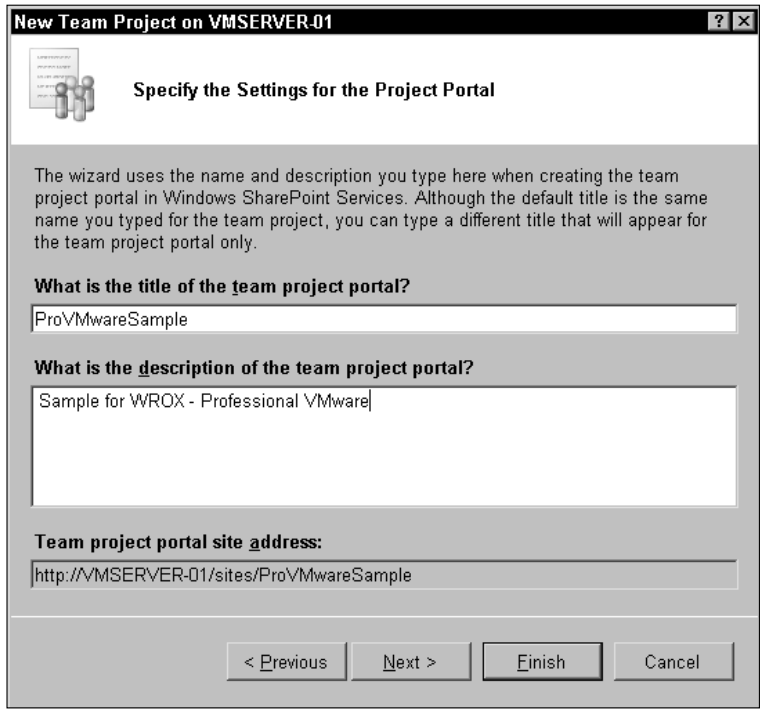
New Team Project on VMSERVER-01
Specify the Settings for the Project Portal
The wizard uses the name and description you type here when creating the team project portal in Windows SharePoint Services. Although the default title is the same name you typed for the team project, you can type a different title that will appear for the team project portal only.
What is the title of the team project portal?
ProVMwareSample
What is the description of the team project portal?
Sample for WROX - Professional VMware
Team project portal site address:
http://VMSERVER-01/sites/ProVMwareSample
< Previous
Next >
Finish
Cancel

Figure 11-24

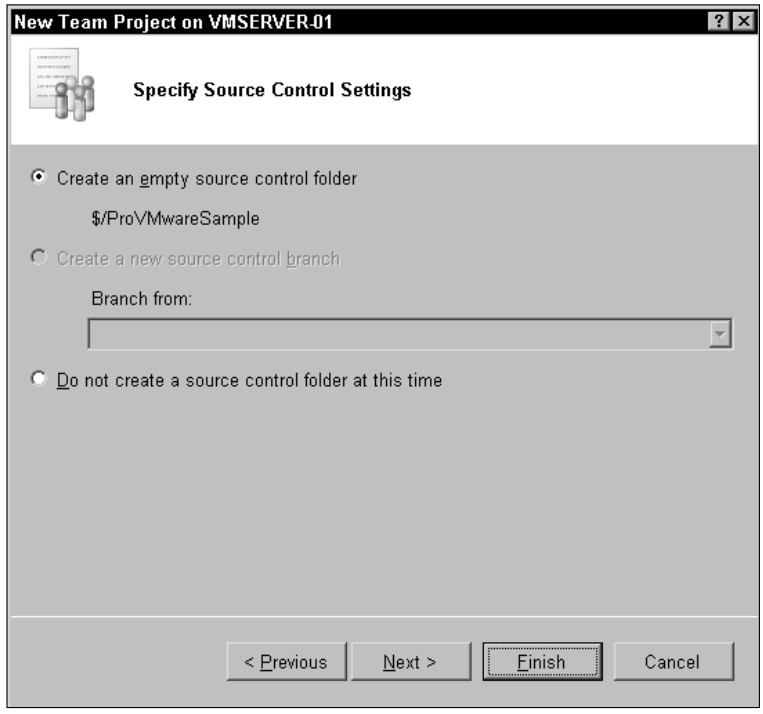
New Team Project on VMSERVER-01
Specify Source Control Settings
Create an empty source control folder
$/ProVMwareSample
Create a new source control branch
Branch from:
Do not create a source control folder at this time
< Previous
Next >
Finish
Cancel

Figure 11-25

12. This brings you to Figure 11-26 and the last step of the wizard.

    Take the plunge and click Finished. You should probably prepare yourself for a bit of a wait, however, because configuring a new project takes a bit of time. You can, of course, just read on and let it catch up with you.

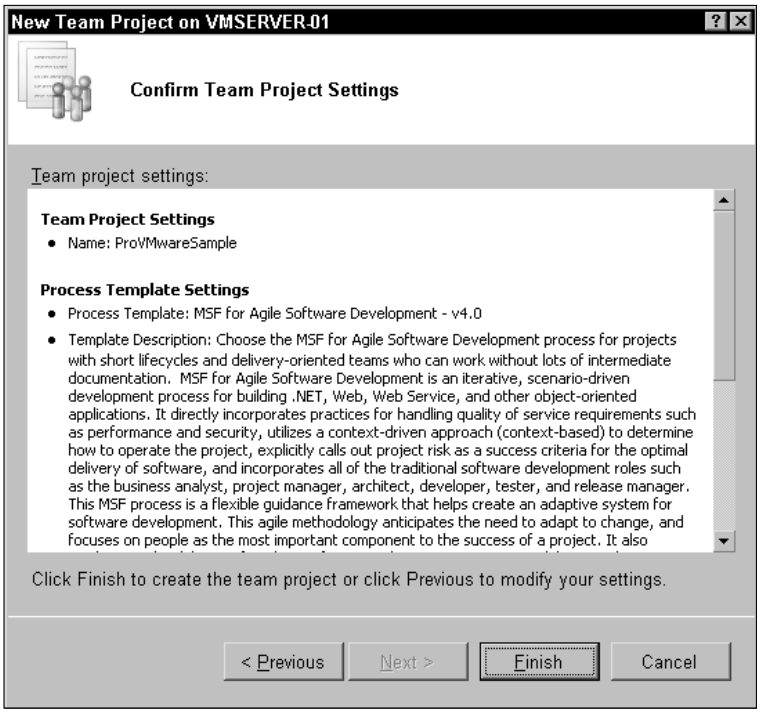
New Team Project on VMSERVER-01
Confirm Team Project Settings
Team project settings:
Team Project Settings
Name: ProVMwareSample
Process Template Settings
Process Template: MSF for Agile Software Development - v4.0
Template Description: Choose the MSF for Agile Software Development process for projects with short lifecycles and delivery-oriented teams who can work without lots of intermediate documentation. MSF for Agile Software Development is an iterative, scenario-driven development process for building .NET, Web, Web Service, and other object-oriented applications. It directly incorporates practices for handling quality of service requirements such as performance and security, utilizes a context-driven approach (context-based) to determine how to operate the project, explicitly calls out project risk as a success criteria for the optimal delivery of software, and incorporates all of the traditional software development roles such as the business analyst, project manager, architect, developer, tester, and release manager. This MSF process is a flexible guidance framework that helps create an adaptive system for software development. This agile methodology anticipates the need to adapt to change, and focuses on people as the most important component to the success of a project. It also
Click Finish to create the team project or click Previous to modify your settings.
< Previous
Next >
Finish
Cancel

Figure 11-26

Now that you have created a new team project you should see something strikingly similar to Figure 11-27. The Team Explorer window is now packed full of several different items. I am not going to dig through all of them here, but feel free to explore them at your leisure. TFS offers a great deal of functionality for team collaboration and almost all of it is exposed through this pane.

Adding a Development Project to the Team Project

Okay, you have your team project but no actual project lives there, so you are now going to take care of that. Use File➪New or any of the other methods Visual Studio gives you to create a new project. It doesn't matter what you create or in what language. The only thing that is important is that it compiles successfully. For this example I just created a new C# WinForms application with a single button on it. Once you are done make sure it builds. Then, it's time to make it a part of your TFS Team Project.

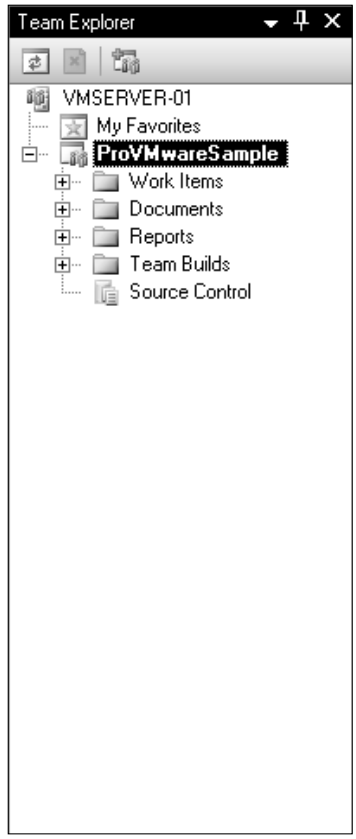

Figure 11-27

The following assumes you have Team Foundation Server set up as your default source control provider. If you do not or wish to verify it, the settings can be found under Tools➪Options in the Source Control node. You may have to reveal all settings under Options to see the Source Control node.

1. As long as you have a source control provider selected under Options, the File menu should have a Source Control submenu. Click File➪Source Control to reveal the menu shown in Figure 11-28. This is how you're going to add the project to the Team Project.

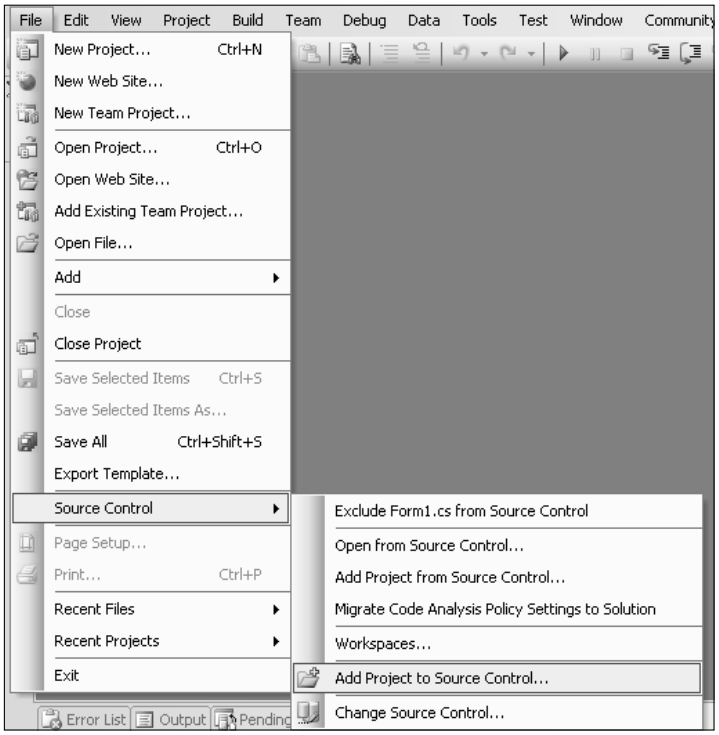

Figure 11-28

2. Select the Add Project to Source Control item to give you the dialog box shown in Figure 11-29. This dialog uses your existing connection to the Team Foundation Server to provide you with a location, or list of locations, to store your code into source control. For now, because you have only one temp project, just accept the defaults and click OK. If you are testing this on a server with more than one project already configured, you may have to make a selection here as to where you want the project to be stored.

3. After a few seconds the IDE will return to normal; however, if you aren't familiar with how Visual Studio identifies files under source control, you'll notice some little plus icons next to the project files. You aren't done yet because the files haven't actually been placed into source control just yet. The little plus icons mean you have the file checked out. Along the bottom of your IDE you should have a tab labeled Pending Changes. Click this tab to reveal the pane shown in Figure 11-30.

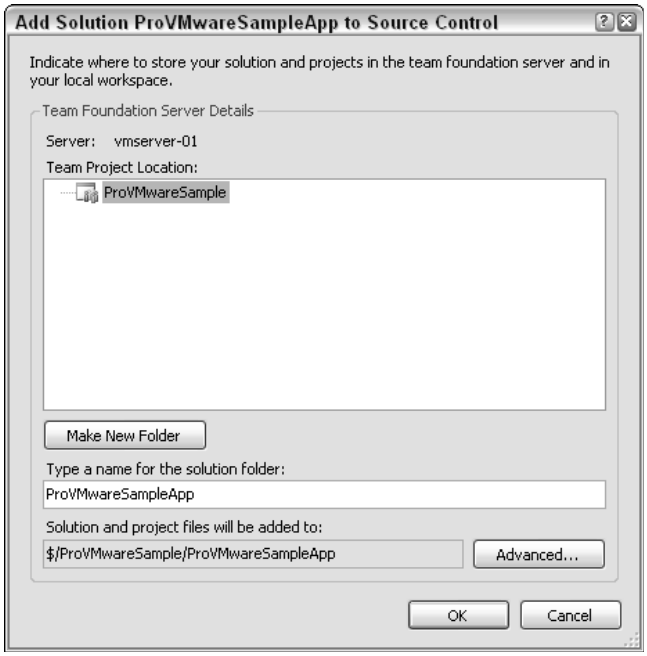
Add Solution ProVMwareSampleApp to Source Control
Indicate where to store your solution and projects in the team foundation server and in your local workspace.
Team Foundation Server Details
Server: vmserver-01
Team Project Location:
ProVMwareSample
Make New Folder
Type a name for the solution folder:
ProVMwareSampleApp
Solution and project files will be added to:
$/ProVMwareSample/ProVMwareSampleApp
Advanced...
OK
Cancel

Figure 11-29

If the Pending Changes tab is not visible along the bottom of your IDE select View➪Other Windows➪Pending Changes to reveal the pane in your IDE.

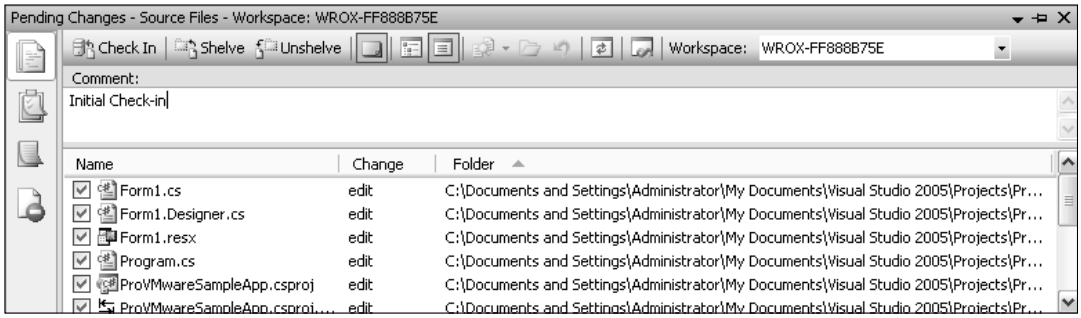
Pending Changes - Source Files - Workspace: WROX-FF888B75E
Check In
Shelve
Unshelve
Workspace: WROX-FF888B75E
Comment:
Initial Check-in
Name
Change
Folder
Form1.cs edit C:\Documents and Settings\Administrator\My Documents\Visual Studio 2005\Projects\Pr...
Form1.Designer.cs edit C:\Documents and Settings\Administrator\My Documents\Visual Studio 2005\Projects\Pr...
Form1.resx edit C:\Documents and Settings\Administrator\My Documents\Visual Studio 2005\Projects\Pr...
Program.cs edit C:\Documents and Settings\Administrator\My Documents\Visual Studio 2005\Projects\Pr...
ProVMwareSampleApp.csproj edit C:\Documents and Settings\Administrator\My Documents\Visual Studio 2005\Projects\Pr...

Figure 11-30

4. The files listed in the lower half of this pane are waiting to be checked into source control. Click the Check In button in the upper left of this pane when you are ready to check the project in. Once this is done, click the Team Explorer tab to view the team project once again. Now, click on Source Control to show the Source Control Explorer for the team project as shown in Figure 11-31.

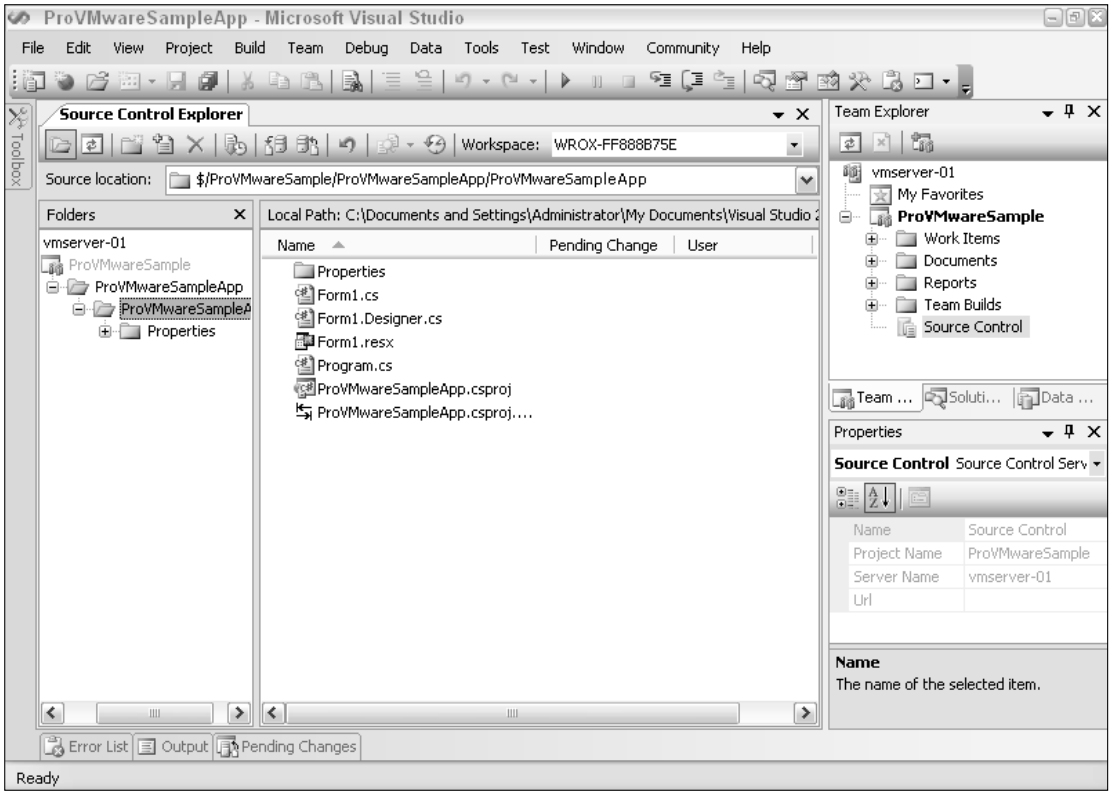

Figure 11-31

You are now a proud owner of a development project within your team project. The sad news is that TFS could care less. It is going to do absolutely nothing other than store this project at the moment. It's high time you change that.

Configuring a Team Build

If you look in the Team Explorer tab you'll see a folder there called Team Builds. Click that folder to expand it, and it will have a whopping one item in it, All Build Types. Technically this is just a filter for the Build Explorer. Go ahead and double-click the All Build Types item and take a look at the Build Explorer. It's pretty empty, but you are going to change that.

1. Move back now to the Team Explorer pane, right-click the Team Builds folder, and select New Team Build Type. This will open the wizard shown in Figure 11-32.

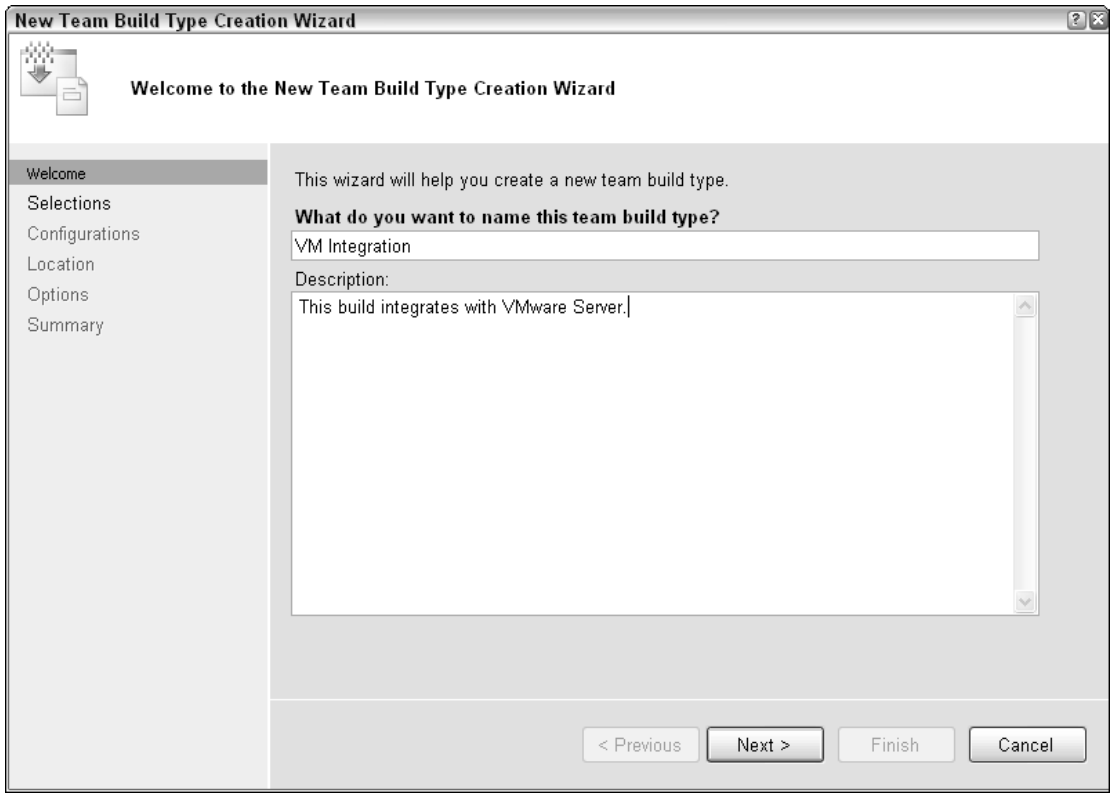

Figure 11-32

As you can tell from the preceding figure, you should name this build VM Integration. Click Next to continue to Figure 11-33, project selection.

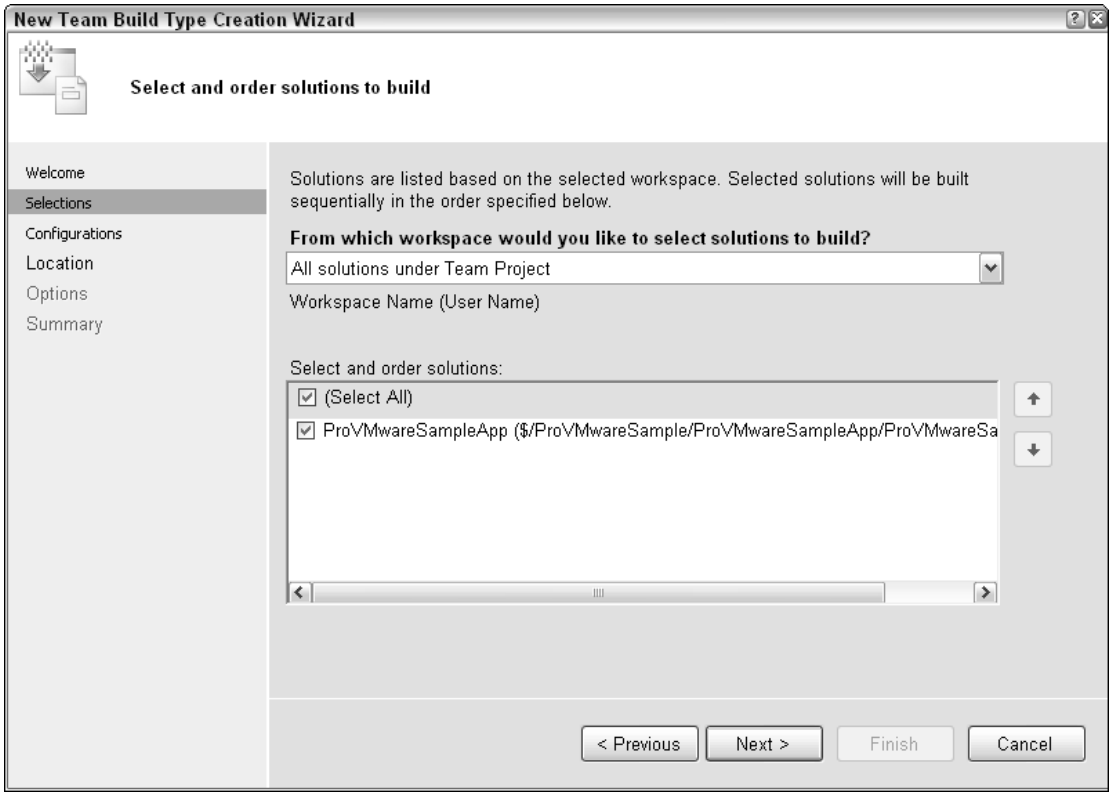

Figure 11-33

2. Here you are given a list of the available projects to build under this team project. Because you just have one project checked in, the list is short. I've selected All solutions under Team Project; however, feel free to choose whichever suits your needs. Once that is complete, click Next.

3. Next, you select the project's build type, as seen in Figure 11-34.

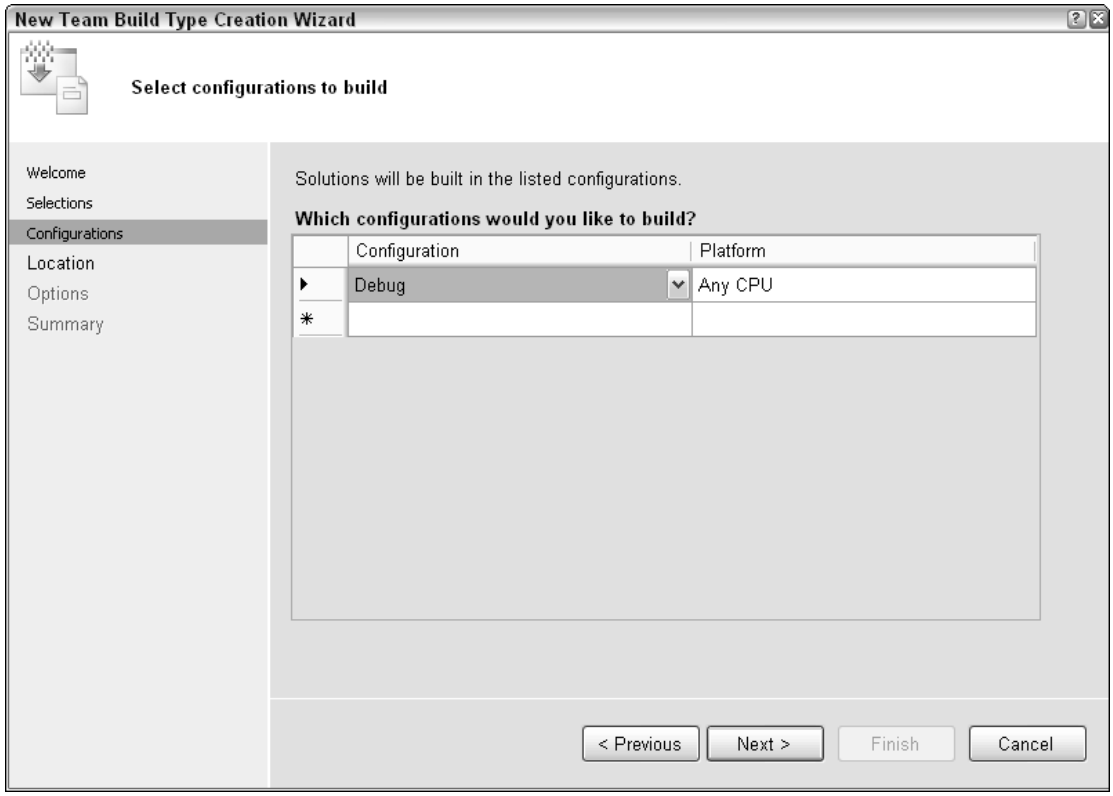

Figure 11-34

Under Configuration you'll notice I selected Debug. The selection really depends on your environment and the stage the code you are testing is in. If you are testing release candidate code, a better selection might be Release; however for this example you are working with production code, so Debug is the better choice. Click Next.

4. Now, you start getting into some guts of this process. Figure 11-35 shows you the build location screen. There are a couple of caveats to this screen that I want to point out.

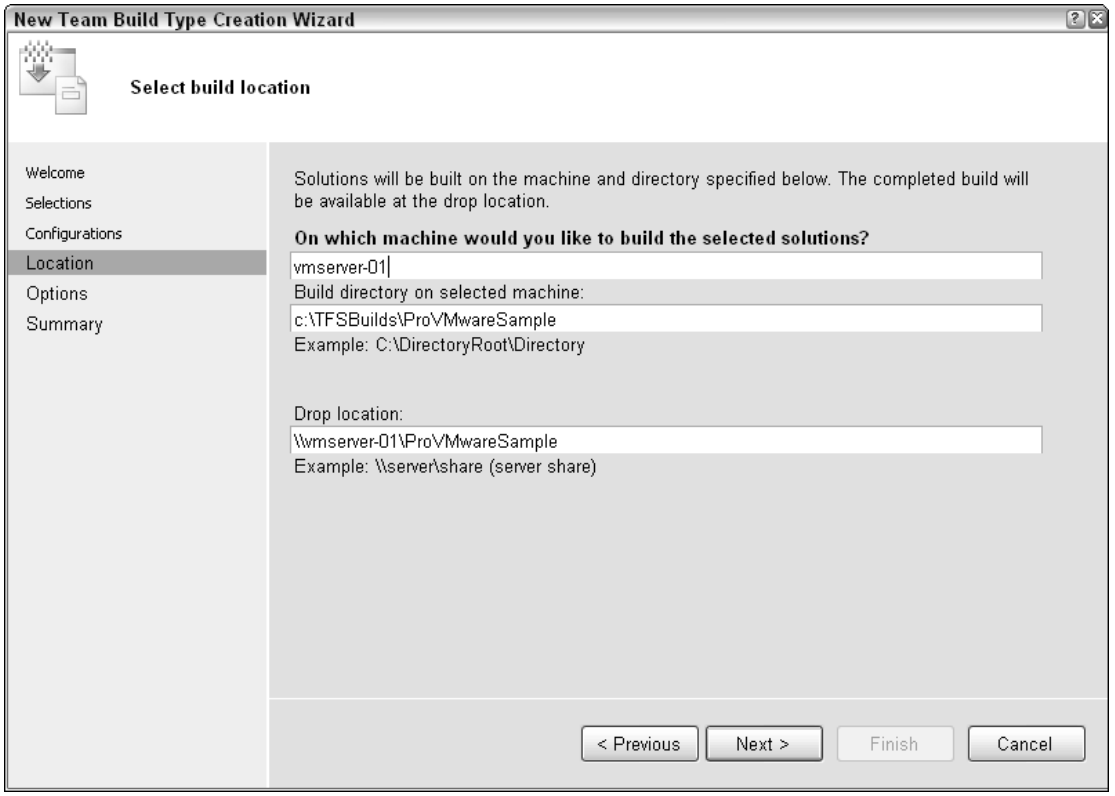

Figure 11-35

Right off the wizard wants to know the name of the machine to perform the build on. Unless you have a separate build machine, this answer is most likely your TFS server. The next two fields ask for a location, relative to the build server, and a share location for the build files. The catch here is that you have to manually create these directories and shares before you kick off this build. You can enter what you like now; the wizard doesn't check for their existence. So, before you use this build, you must go create them or the build will fail. Make sure you apply the appropriate security permissions to the share. The automated builds are run under the user context the Team Build service runs under. The TFS install documentation refers to this account as the TFSService account. When you are finished, click Next.

5. You now travel onto the unit test and code analysis setup screen, as shown in Figure 11-36.

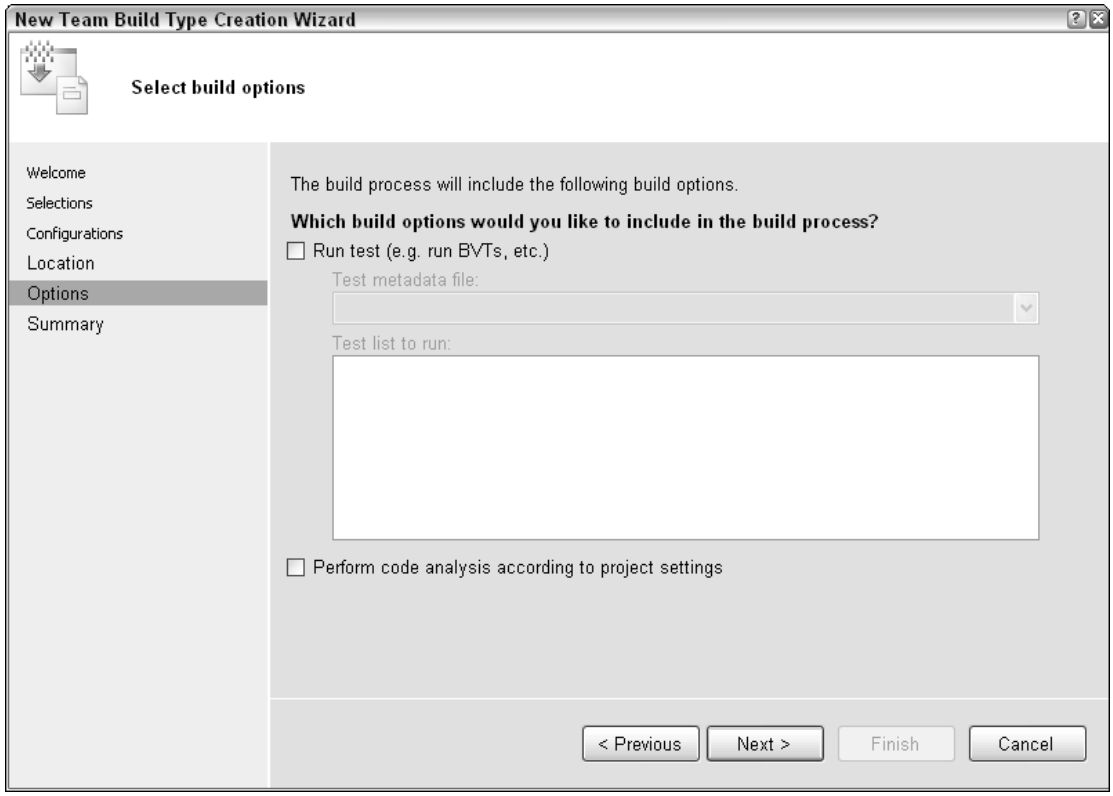

Figure 11-36

This is the final step of the wizard. If you have any Microsoft Testing Framework tests embedded into your project, this is where you would select the metadata file associated with them. Because that isn't the case for this example, the checkbox is left blank.

The second option is to perform code analysis on the project at build. This refers to code coverage tools and so forth that are built into the Team Editions of Visual Studio. For now it's not a concern. Click Next for the final screen of the wizard.

6. You are now faced with a simple review screen before you finish. If everything looks good go ahead and click Finish.

You should now have a new build type under the Team Explorer Team Build folder called VM Integration, as shown in Figure 11-37.

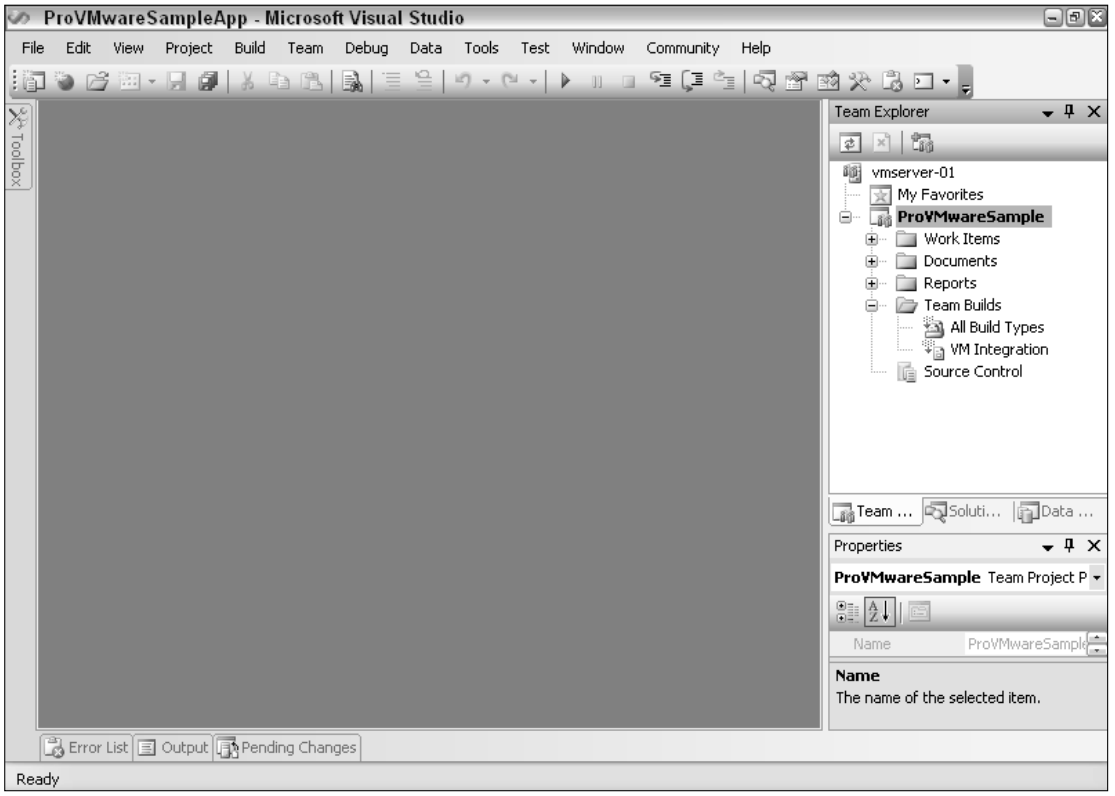

Figure 11-37

Testing the Build

Now it's time to test out your build:

1. Right-click the VM Integration build under Team Builds and choose Build Team Build Project ProVMwareSampleApp. This brings up the build screen shown in Figure 11-38. This just allows some last minute adjustments to how you want the build to proceed. If you configured the build type correctly and created the appropriate directories and shares on the TFS server, you will not need to change these values.
2. If you are satisfied with the setup go ahead and click Build. The process will take several seconds and greet you with a nice progress bar while you wait. Regardless of the build outcome, you will ultimately end up at Figure 11-39, the build log window.

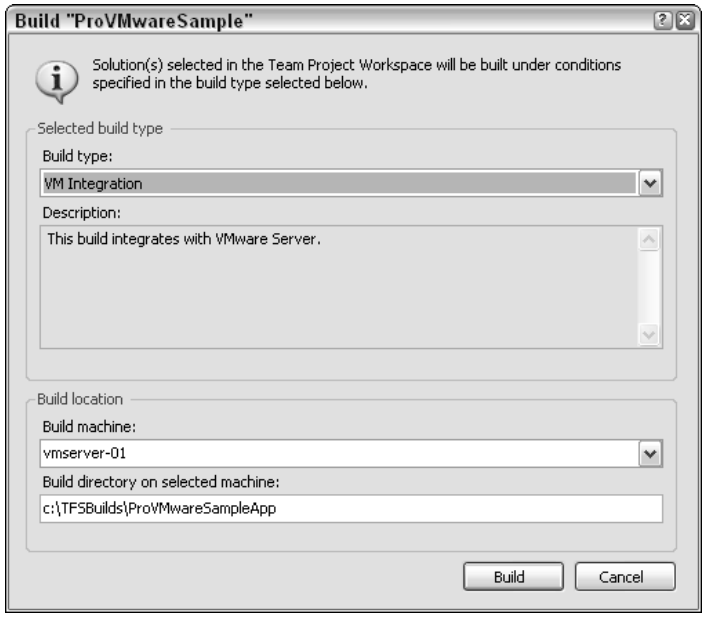

Figure 11-38

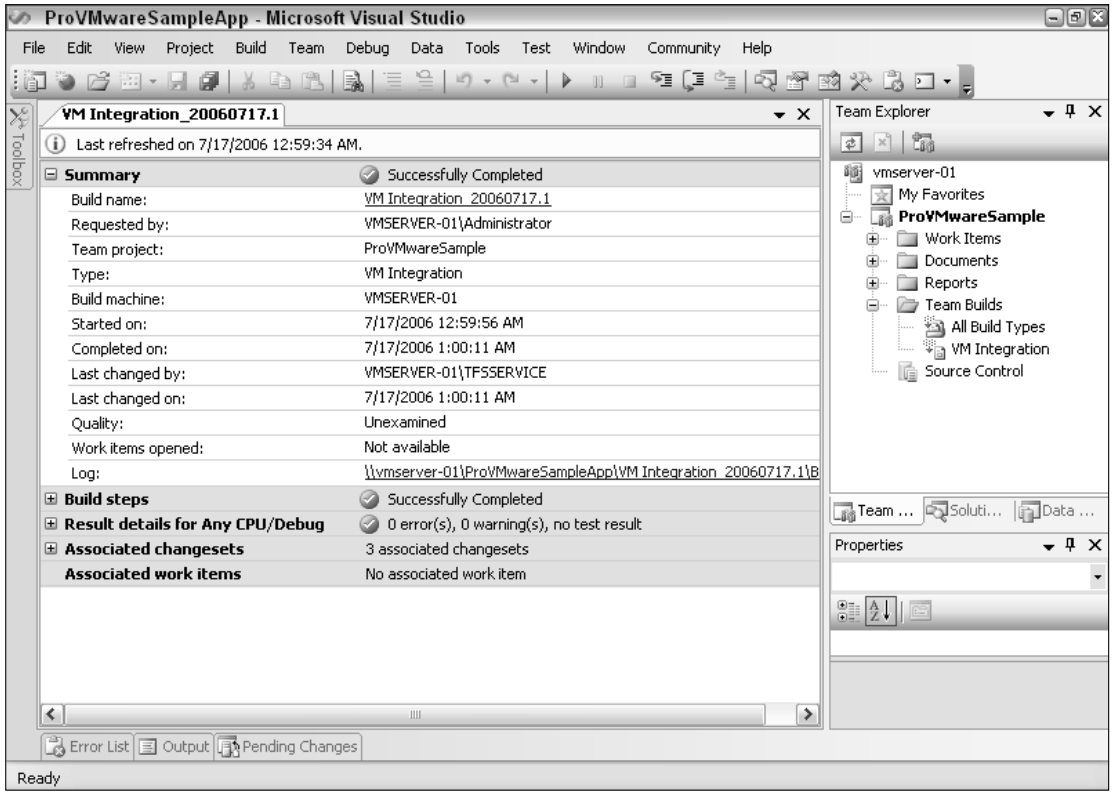

Figure 11-39

If you have green checkmarks, then congratulations — you just completed perhaps your first TFS build. If you do not have a successful build, go back and check the project code files for an error. In this scenario you are just using a blank project, which will compile if left unchanged; however, you may be using your own code for the example. If that is the case check your code for errors or perform a build in the IDE manually to get a list of errors.

You are probably wondering why you called this build VM Integration since nothing integrated. Well, that's where you are heading next.

VMware Server Integration

Team Foundation Server doesn't really provide any easy way to integrate a script into the build that will fire up your virtual machines like CruiseControl.NET did. In fact, CCNet was incredibly easy compared to what's in store for you next.

TFS provides several hooks into the build process. You can literally place a task into almost any stage of the build. Of course you want to call your virtual machine up after the build and testing has taken place, so after reviewing the TFS documentation, the best hook for you will be either `BeforeDropBuild` or `AfterDropBuild`. These hooks are added to the build's XML configuration file, which I cover a bit later in this chapter. The thing to understand here is that, unlike CCNet where you could just make an external call, VSTS requires your build hooks to leverage an assembly. This means that to fire up a virtual machine your custom code has to be handled through a custom developed assembly. However, this is not a problem because you are going to whip one up.

Creating a Custom Assembly for Integration

The following walks you through creating a new class library project that you will use to hook into the build process within VSTS:

1. Open your IDE back up and start a new class library project. I will be using C#; however, I will also provide the same code in VB on the book's web site at `www.wrox.com`. Call the project VMIntegration. Do not, however, place this into source control for your team project. You will be doing something special with this project's DLL once it's built.
2. Once you have the new project created, rename the class to `VMwareServer`. Figure 11-40 gives you a jumping off point.

   Something beautiful about this solution is that you no longer have to rely on an external script, as you did with the Perl scripts from before, because you can leverage the VMWare COM API from directly within this class project. However, to do this you need to install the COM API onto your development machine. You can either download the Windows Client package from VMware or simply use a custom install from your server package.

3. You need to add three references to your project to leverage the MSBuild process and VMware API. Right-click References in the Solution Explorer and choose Add Reference. Figure 11-41 should appear with a list of .NET assemblies available to reference.

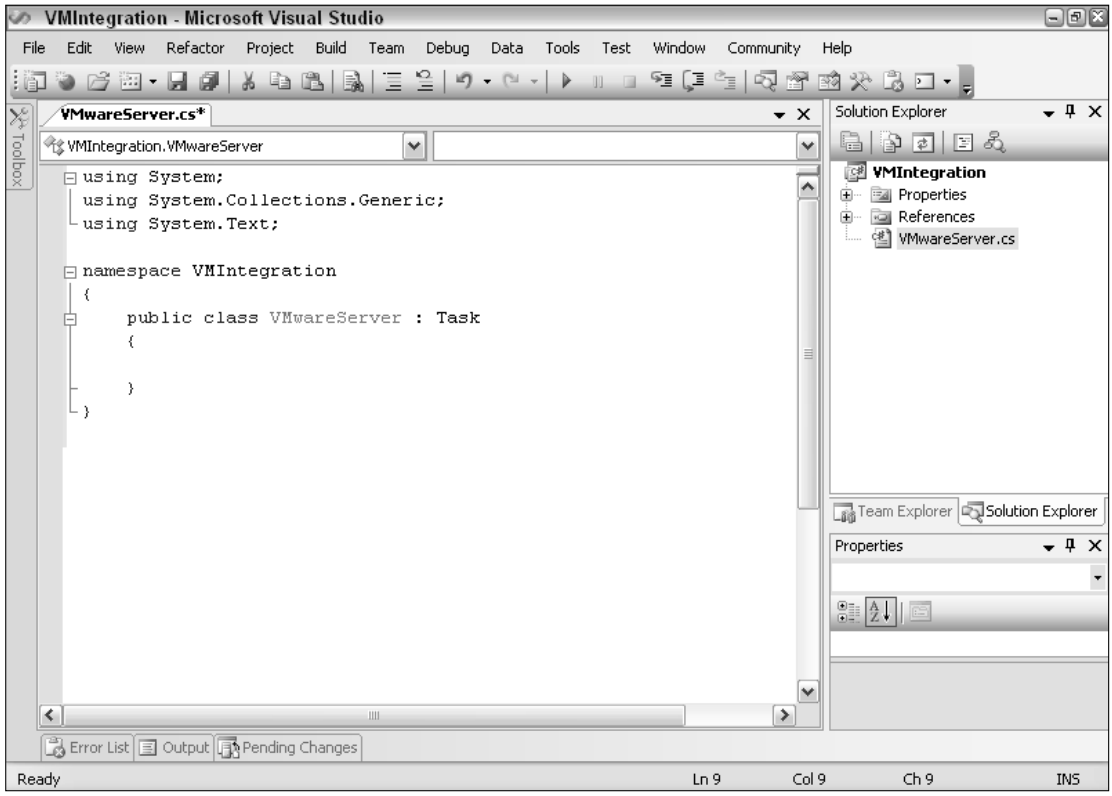

Figure 11-40

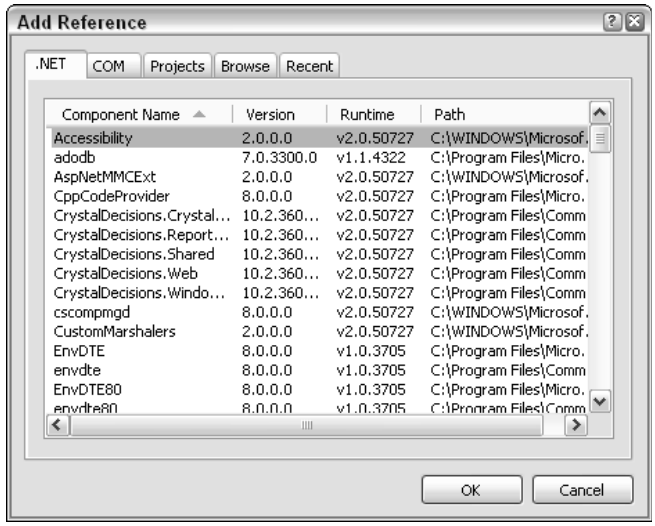

Figure 11-41

4. The two assemblies you need to add references under .NET are `Microsoft.Build.Framework` and `Microsoft.Build.Utilities`. You can either do a Ctrl+click on both to add them at the same time or simply repeat this procedure twice. Regardless, select the assemblies and choose OK.

5. Next, you will want to reopen this dialog, select the COM tab, and choose the VMware VmCOM 1.0 Type Library from the list. If you added them correctly, you should be able to expand your References folder in the Solution Explorer and see the three newly added items. Figure 11-42 shows you what it should look like.

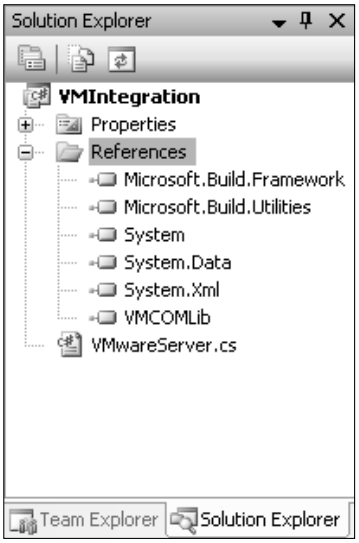

Figure 11-42

6. With this done, you can shift your focus back to the task at hand. Make sure the `VMwareServer.cs` class file is open in the editor and remove everything from within the file. Add the following to the `VMwareServer.cs` class file exactly as shown. As usual, I will show the entire class at the end of this section.

```
using System;
using System.Collections.Generic;
using System.Text;
using System.Text.RegularExpressions;
using Microsoft.Build.Framework;
using Microsoft.Build.Utilities;
using VMCOMLib;

namespace VMIntegration
{
    public class VMwareServer : Task
    {

    }
}
```

The reason I had you do this is to ensure a couple requirements were met:

- First, you need the new `using` declarations at the top in order to utilize the new Build assemblies and VmCOM API you added as references.
- Second, and almost as important, the `VMwareServer` class must inherit from the `Task` class in order for MSBuild to be able to use it as such. This code block accomplishes both for you.

7. Now, the sole requirement of implementing a class that inherits from `Task` is that you have a method named `Execute()`. To satisfy this insert the following code into the `VMwareServer` class block:

```
public override bool Execute()
{
    return true;
}
```

Coding Your Build Task

At this point, technically, the assembly if integrated into the build project would work perfectly. The requirements are met, and when executed, it simply returns true. From that statement alone you can probably deduce that you need to add your functionality into this `Execute()` method.

The way that you hook into the team project XML file allows you to pass some information over to your assembly. You are going to take advantage of that. The following list shows the information you need to get passed in:

- The build directory
- The virtual machine name pattern you need for this project (`Regex` expression)
- The VMware Server name, port, user name, and password to use

This information is passed into your assembly using properties that you have to set up. The other end of this connection, the XML file, simply contains some extra attributes that I cover in a bit. For now, you can add the properties you need. These, like before, go within the class definition block:

```
private string sourceDir;
private string serverName;
private string serverPort;
private string username;
private string password;
private string namePattern;

[Required]
public string SourceDir
{
  get { return sourceDir; }
  set { sourceDir = value; }
}

public string ServerName
{
```

```
    get { return serverName; }
    set { serverName = value; }
  }

  public string ServerPort
  {
    get { return serverPort; }
    set { serverPort = value; }
  }

  public string UserName
  {
    get { return username; }
    set { username = value; }
  }

  public string Password
  {
    set { password = value; }
  }

  [Required]
  public string NamePattern
  {
    get { return namePattern; }
    set { namePattern = value; }
  }
```

You'll notice the `[Required]` metadata tag on two of the properties. This means that you at least have to send the assembly the source directory and a name pattern to match. The rest of the items can be left out in case you are connecting to a local VMware Server and do not need to pass any connection information.

Now, you can start building up the `Execute()` method. Because this is where all the magic will happen, you'll take it a few steps at a time:

```
  string binaryDirectory = SourceDir + "\\..\\Binaries ";

  VmConnectParams vmConn = new VmConnectParams();
  VmServerCtl vmServer = new VmServerCtl();

  // This are set based on our properties above.
  vmConn.Hostname = ServerName;
  vmConn.Port = ServerPort;
  vmConn.Username = UserName;
  vmConn.Password = Password;

  // Attempt to connect to the server and if it fails log to the build log.
  try
  {
    vmServer.Connect(vmConn);
  }
    catch (Exception ex)
  {
```

```
  Log.LogMessage("Could not connect to server.");
  return false;
}

// Our two arraylists that hold the vm's that match and are available to use.
ArrayList vmMatching = new ArrayList();
ArrayList vmAvailable = new ArrayList();

// This is where we take the namePattern provided and file our vm's.
Regex vmNamePattern = new Regex(NamePattern);

Log.LogMessage("The following machines were found matching the criteria
                provided.");

foreach (string name in vmServer.RegisteredVmNames)
{
  if (vmNamePattern.IsMatch(name))
  {
  vmMatching.Add(name);
  Log.LogMessage(name);
  }
}

if (vmMatching.Count == 0)
{
  Log.LogMessage("No virtual machines are registered.");
}
```

You start off by defining the location of your binaries for the project and setting up the connection parameters and server object. Next, you connect to the server and retrieve a list of registered virtual machines. Then, this list is compared against the regular expression provided, and the matches are placed into an `ArrayList` named `vmMatching`. Aside from a bit of error handling and logging that's it for this block.

Now, you need to connect to each virtual machine in the `vmMatching ArrayList` and determine which ones are available for use:

```
// Walk through the matching vm's and connect to see what state they are in.
// If they are available for use push them into the available arraylist.
foreach (string name in vmMatching)
{
    VmCtl vm = new VmCtl();

    // Attempt to connect and if it fails log to the build log.
    try
    {
      vm.Connect(vmConn, name);
    }
    catch (exception ex)
   {
      Log.LogMessage("Unable to connect to virtual machine: Status verifier");
      return false;
    }

    VMCOMLib.VmExecutionState status = vm.ExecutionState;
```

```
        // Check the machine's state.  If it is currently OFF then push it into the
        // available arraylist.
        if (status == VMCOMLib.VmExecutionState.vmExecutionState_Off)
        {
            vmAvailable.Add(name);
            Log.LogMessage(name);
        }
    }

    if (vmAvailable.Count == 0)
    {
        Log.LogMessage("No virtual machines match the criteria.");
        return false;
    }
```

Here you simply step through the array list, connect to each machine, and poll its status. If the status of the particular machine is OFF, then you add it to the `vmAvailable arraylist`.

Next, you pluck off one of the available machines and start it up:

```
    VmCtl vmInUse = new VmCtl();

    string vmConfigFileName = vmAvailable[0].ToString();

    Log.LogMessage("The configuration file in use is " + vmConfigFileName);

    // Attempt to connect and if it fails log to the build log.
    try
    {
        vmInUse.Connect(vmConn, vmAvailable[0].ToString());
    }
    catch (exception ex)
    {
        Log.LogMessage("Unable to connect to virtual machine: Startup");
    }

    Log.LogMessage("Starting machine");

    // Attempt to startup machine.  If it fails log to the build log.
    try
    {
    vmInUse.Start(VmPowerOpMode.vmPowerOpMode_Soft);

      Log.LogMessage("Sleeping for 45 seconds...");
      System.Threading.Thread.Sleep(45000);
    }
    catch (Exception ex)
    {
      Log.LogMessage("Unable to start virtual machine: Startup");
      return false;
    }

    //Now that the machine is hopefully running grap the IP address from the
    // guestinfo.ip variable.
```

```
string vmIPAddress = vmInUse.get_GuestInfo("ip");

Log.LogMessage("The vm's IP addres is " + vmIPAddress);

return true;
```

So, you take the first available virtual machine and start it up. Then, in order to give the guest time to get up and running, you put the thread to sleep for 45 seconds or so. After that you grab the IP address from the VMware Tools script to be used during the file copy.

Copying Binaries to the Virtual Machine

Now, you turn into the home stretch, copying the binaries up to the test machine. There are several ways in which you can do this. The caveats here are that the virtual machine is a completely separate beast from the rest of the network machines. This means authentication is required to copy the files. In the CCNet example earlier in the chapter I used three external calls in order to accomplish this. I am going to do the same here for two reasons:

- The first is that the external call, which will be a batch file, is easily configurable. That way if you change where you want the files dropped, then all you have to do is edit the batch file. No need to open the IDE and recompile your library.
- The second reason is network security. For you to make this file copy from within .NET, you have to play around with security tokens and network authentication. This isn't the easiest thing to do. You would need to impersonate a valid user on the guest machine, copy the files under that context, and then revert back to the original token. Because the guest machines you will be connecting to can vary from operating system to the local users configured on them I think the batch file solution is the best choice for this example. It allows a degree of flexibility among guest machines and keeps you from recompiling the library if things change.

Open a text editor and enter the following. Save this file as `vmCopyFiles.bat` and remember where you put it. You will be checking it in with the DLL in a bit.

```
net use x: "\\%1\c$\Documents and Settings\Administrator\Desktop"
/user:administrator Pass@word!

xcopy %2 x:\Binaries\

net use x: /delete
```

Finally, you need to finish up the `VMIntegration` class. Insert the following code just above the final `return true;` statement:

```
vmCopyFiles(vmIPAddress, binaryDirectory);
```

Next, you need to add the method this calls. After the `Execute()` method insert the following:

```
private void vmCopyFiles(string computerIP, string binaryDirectory)
{
    System.Diagnostics.ProcessStartInfo proc = new
    System.Diagnostics.ProcessStartInfo(sourceDir + "\\tools\\vmCopyFiles.bat");
    proc.Arguments = computerIP + " \"" + binaryDirectory + "\\Debug\\*.*\"";
```

```
        proc.WindowStyle = System.Diagnostics.ProcessWindowStyle.Hidden;
        proc.UseShellExecute = true;

        System.Diagnostics.Process copyFiles;

        copyFiles = System.Diagnostics.Process.Start(proc);

        copyFiles.WaitForExit(5000);
    }
```

This last method handles the actual copy. As the final state of `Execute()`, you call this method passing the guest machine's IP address and the `binaryDirectory` of the project. With these two pieces of information you spawn a new process and run your batch file `vmCopyFiles.bat` passing them as arguments. The batch file then takes over by creating a new mapped network drive, copying the files, and then removing the drive letter. It may seem a little archaic to use a batch file, but hey, it gets the job done.

The Final VMIntegration Class

As promised I will post the class in its entirety. You aren't quite done with it but the edits to the code itself are complete.

VMIntegration.cs

```
using System;
using System.Collections;
using System.Text;
using System.Text.RegularExpressions;
using Microsoft.Build.Framework;
using Microsoft.Build.Utilities;
using VMCOMLib;

namespace VMIntegration
{
    public class VMwareServer : Task
    {
        private string sourceDir;
        private string serverName;
        private int serverPort;
        private string username;
        private string password;
        private string namePattern;

        #region "Properties"

        [Required]
        public string SourceDir
        {
            get { return sourceDir; }
            set { sourceDir = value; }
        }

        public string ServerName
        {
```

```
            get { return serverName; }
            set { serverName = value; }
        }

        public int ServerPort
        {
            get { return serverPort; }
            set { serverPort = value; }
        }

        public string UserName
        {
            get { return username; }
            set { username = value; }
        }

        public string Password
        {
            get { return password; }
            set { password = value; }
        }

        [Required]
        public string NamePattern
        {
            get { return namePattern; }
            set { namePattern = value; }
        }

        #endregion

        public override bool Execute()
        {
            string binaryDirectory = SourceDir + "\\..\\Binaries";

            VmConnectParams vmConn = new VmConnectParams();
            VmServerCtl vmServer = new VmServerCtl();

            // These are set based on our properties above.
            vmConn.Hostname = ServerName;
            vmConn.Port = ServerPort;
            vmConn.Username = UserName;
            vmConn.Password = Password;

            // Attempt to connect to the server and if it fails log to the build
            //log.
            try
            {
                vmServer.Connect(vmConn);
            }
            catch (Exception ex)
            {
                Log.LogMessage("Could not connect to server.");
                return false;
```

(continued)

VMIntegration.cs *(continued)*

```
        }

        // Our two arraylists that hold the vm's that match and are available
        //to use.
        ArrayList vmMatching = new ArrayList();
        ArrayList vmAvailable = new ArrayList();

        // This is where we take the namePattern provided and file our vm's.
        Regex vmNamePattern = new Regex(NamePattern);

        Log.LogMessage("The following machines were found matching the ↵
                       criteria provided.");
        foreach (string name in vmServer.RegisteredVmNames)
        {
            if (vmNamePattern.IsMatch(name))
            {
                vmMatching.Add(name);
                Log.LogMessage(name);
            }
        }

        if (vmMatching.Count == 0)
        {
            Log.LogMessage("No virtual machines are registered.");
        }

        // Walk through the matching vm's and connect to see what state they
        //are in.
        // If they are available for use push them into the available
        //arraylist.

        foreach (string name in vmMatching)
        {
            VmCtl vm = new VmCtl();

            // Attempt to connect and if it fails log to the build log.
            try
            {
                vm.Connect(vmConn, name);
            }
            catch (Exception ex)
            {
                Log.LogMessage("Unable to connect to virtual machine: ↵
                               Status verifier");
            }

            VMCOMLib.VmExecutionState status = vm.ExecutionState;

            // Check the machine's state.  If it is currently OFF then push it
            //into the available arraylist.
            if (status == VMCOMLib.VmExecutionState.vmExecutionState_Off)
            {
                vmAvailable.Add(name);
```

```
                    Log.LogMessage(name);
                }
            }

            if (vmAvailable.Count == 0)
            {
                Log.LogMessage("No virtual machines match the criteria.");
                return false;
            }

            VmCtl vmInUse = new VmCtl();

            string vmConfigFileName = vmAvailable[0].ToString();

            Log.LogMessage("The configuration file in use is " + vmConfigFileName);

            // Attempt to connect and if it fails log to the build log.
            try
            {
                vmInUse.Connect(vmConn, vmAvailable[0].ToString());
            }
            catch (Exception ex)
            {
                Log.LogMessage("Unable to connect to virtual machine: Startup");
                return false;
            }

            Log.LogMessage("Starting machine");

            // Attempt to startup machine.  If it fails log to the build log.
            try
            {
                vmInUse.Start(VmPowerOpMode.vmPowerOpMode_Soft);

                Log.LogMessage("Sleeping for 45 seconds...");
                System.Threading.Thread.Sleep(45000);
            }
            catch (Exception ex)
            {
                Log.LogMessage("Unable to start virtual machine: Startup");
                return false;
            }

            //Now that the machine is hopefully running grap the IP address from
            //the guestinfo.ip variable.
            string vmIPAddress = vmInUse.get_GuestInfo("ip");

            Log.LogMessage("The vm's IP addres is " + vmIPAddress);

            vmCopyFiles(vmIPAddress, binaryDirectory);

            return true;
        }

        private void vmCopyFiles(string computerIP, string binaryDirectory)
```

(continued)

VMIntegration.cs *(continued)*

```
        {
            System.Diagnostics.ProcessStartInfo proc = new
             System.Diagnostics.ProcessStartInfo(sourceDir +
             "\\tools\\vmCopyFiles.bat");
            proc.Arguments = computerIP + " \"" + binaryDirectory +
             "\\Debug\\*.*\"";

            proc.WindowStyle = System.Diagnostics.ProcessWindowStyle.Hidden;
            proc.UseShellExecute = true;

            System.Diagnostics.Process copyFiles;

            copyFiles = System.Diagnostics.Process.Start(proc);

            copyFiles.WaitForExit(5000);
        }
    }
}
```

At this point, you have two things left to do:

- Building and placing this class library and batch file into your team project
- Setting up the team build's XML file

Placing the Class Library and Batch File in Your Team Project

You can start these final tasks off with the build, so go ahead and give the VMIntegration library one final build just to wrap things up. Once that is complete you need to add it to our team project's source control. Just the .DLL file is all you need to add, not the whole project.

> *For the final build you want to set the build configuration to Release instead of the default, which is Debug. Select Build➪Configuration Manager from the menu. Once the dialog is open, change the configuration to Release for the VMIntegration project and click Close.*

1. Open up your project directory under My Documents and navigate into the `bin/release` folder, as shown in Figure 11-43.

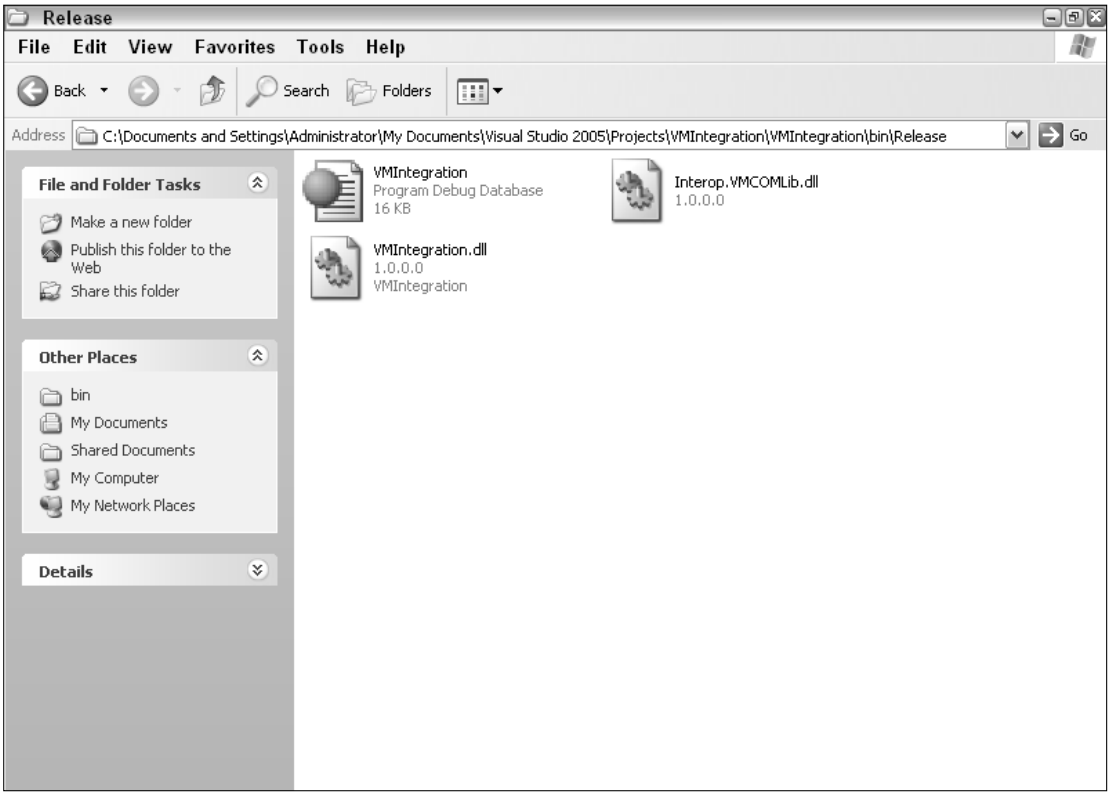

Figure 11-43

2. Once there you copy the two DLL files, `VMIntegration.dll` and `Interop.VMCOMLib.DLL`.
3. Next, back up out of the project directory into the team project directory, ProVMwareSampleApp, shown in Figure 11-44.

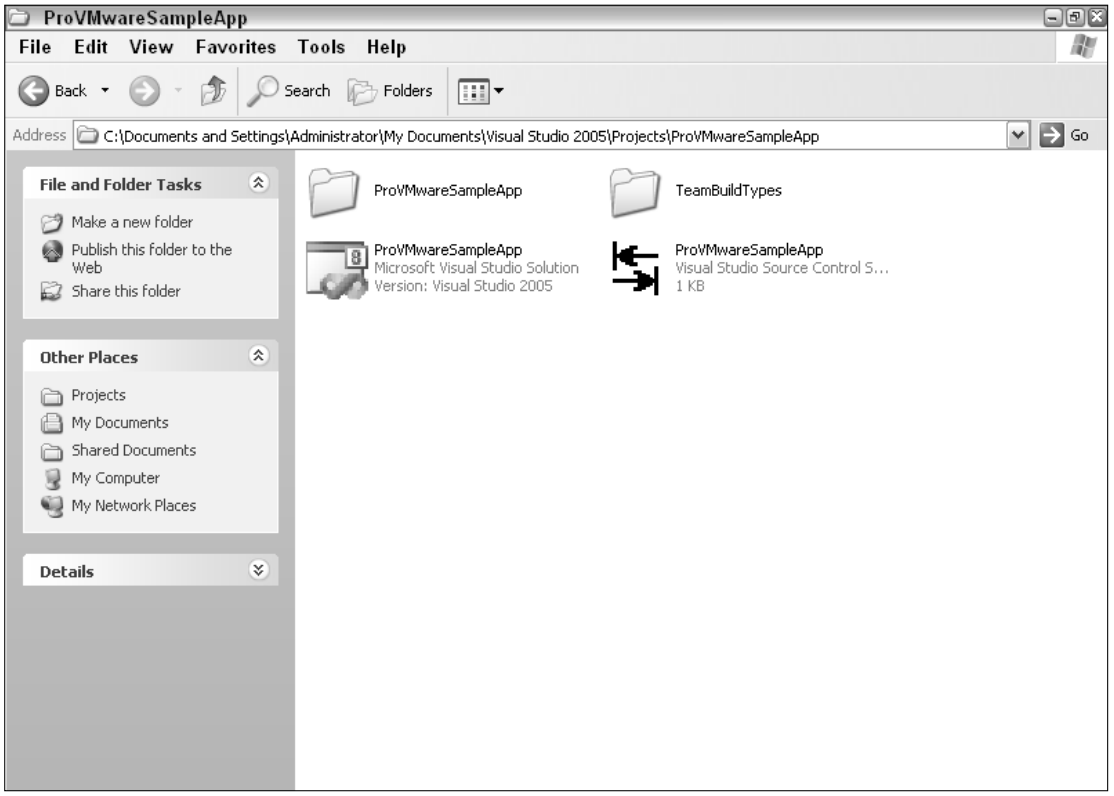

Figure 11-44

4. Create a new folder here called tools and place the two DLL files there. Also, locate the batch file you created along with the integration library and copy it there as well. The result should look like Figure 11-45.

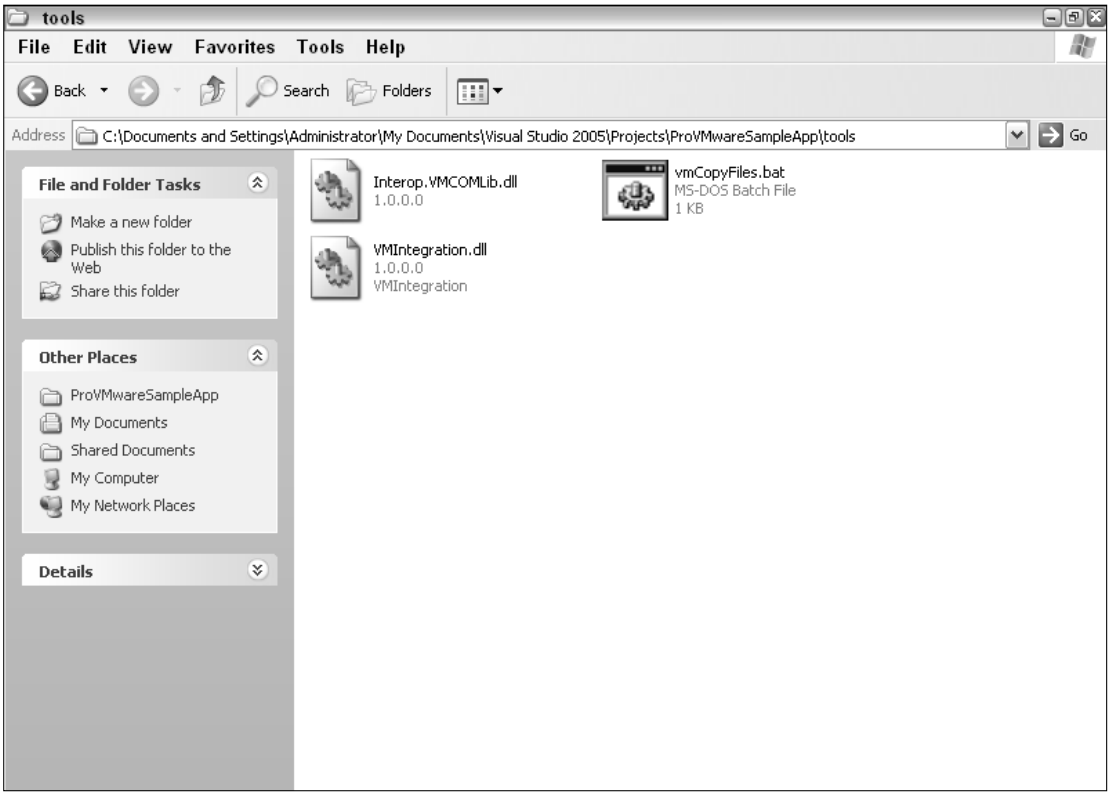

Figure 11-45

5. You need to add this new folder into the team project's source control. Open the Visual Studio IDE and from the Team Explorer double-click the Source Control icon. This should bring up your project's Source Control Explorer, as shown in Figure 11-46.

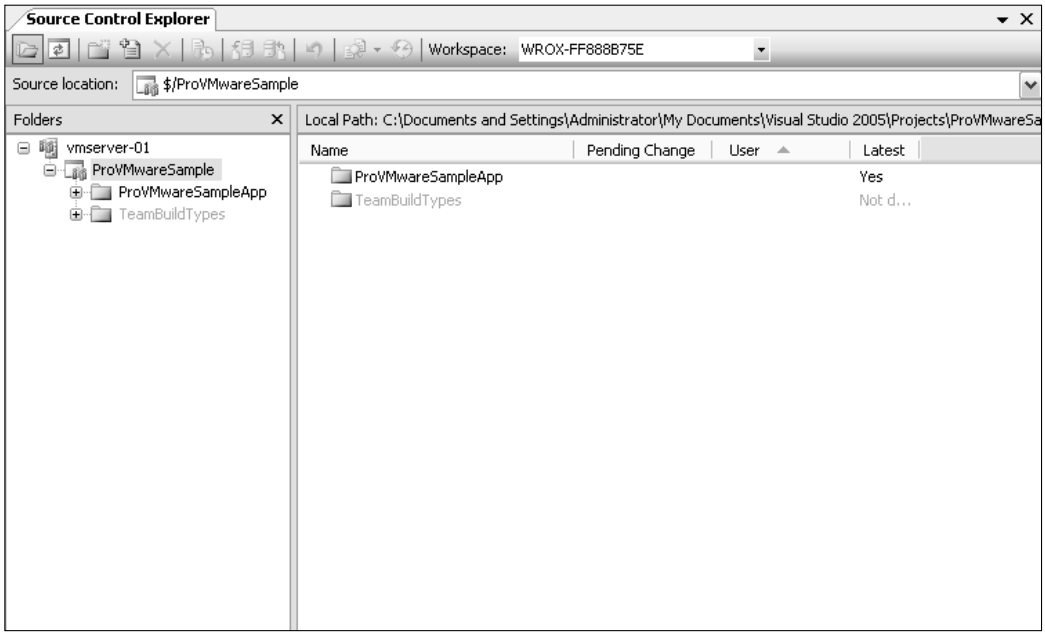

Figure 11-46

6. Now you want to add this folder to the top-most ProVMwareSample node because that's where you placed it in the folder structure. Click the Add Files icon on the toolbar for the Source Control Explorer. It should bring up a dialog similar to Figure 11-47.

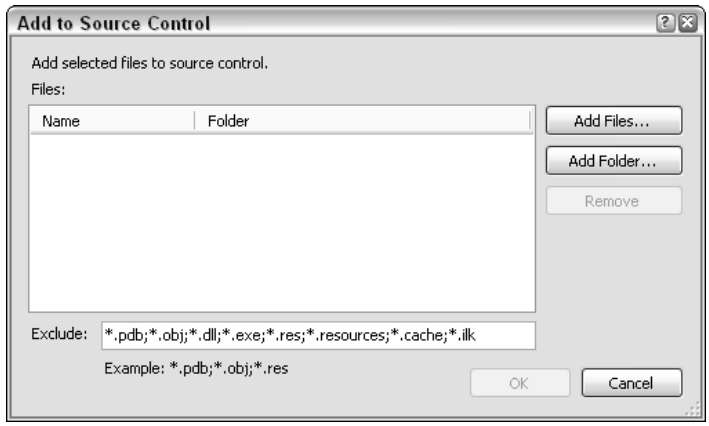

Figure 11-47

7. Because you are adding an entire folder click the Add Folder button. This will bring up the familiar Browse for Folder dialog you've seen before.

8. Navigate to your project folder, select the tools folder, and click OK. This will bring up the select files window as shown in the preceding figure, Figure 11-47; however, now it will contain the three files that you copied into the tools folder. Place a checkmark in all three boxes and click OK.

9. A brief progress bar will appear and your files will be added. But wait — they aren't in the Source Explorer. You still need to commit the changes to the database. Open the Pending Changes tab at the bottom of the IDE, shown in Figure 11-48, and click the Check In button. This commits the changes. You should now be able to see the new tools folder in the Source Explorer. You may, however, have to refresh the screen in order to see them.

If the Pending Changes tab is not visible along the bottom of your IDE, select View➪Other Windows➪Pending Changes to reveal the pane in your IDE.

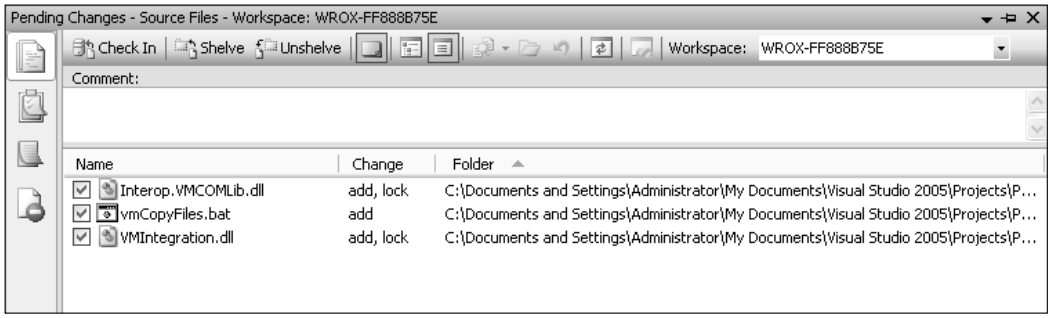

Figure 11-48

You're done with this step. What you've actually accomplished here is to provide the team build with the assembly as part of the project's source so that no matter what machine you build it on the proper assemblies are available to it.

Adding the New Assembly to the Team Build XML File

Last, you need to adjust the team build XML configuration file slightly to use your new assembly:

1. From the Source Control Explorer open the TeamBuildTypes/VM Integration folder. This folder contains the XML file you need to edit. Figure 11-49 shows the folder and its files.

2. Right-click the `TFSBuild.proj` file and choose Check out for Edit. It may prompt you for where to place the file. If it does, point it to the same place you put the tools folder. Once you have it checked out, double-click the file from within the Source Explorer to open it for editing.

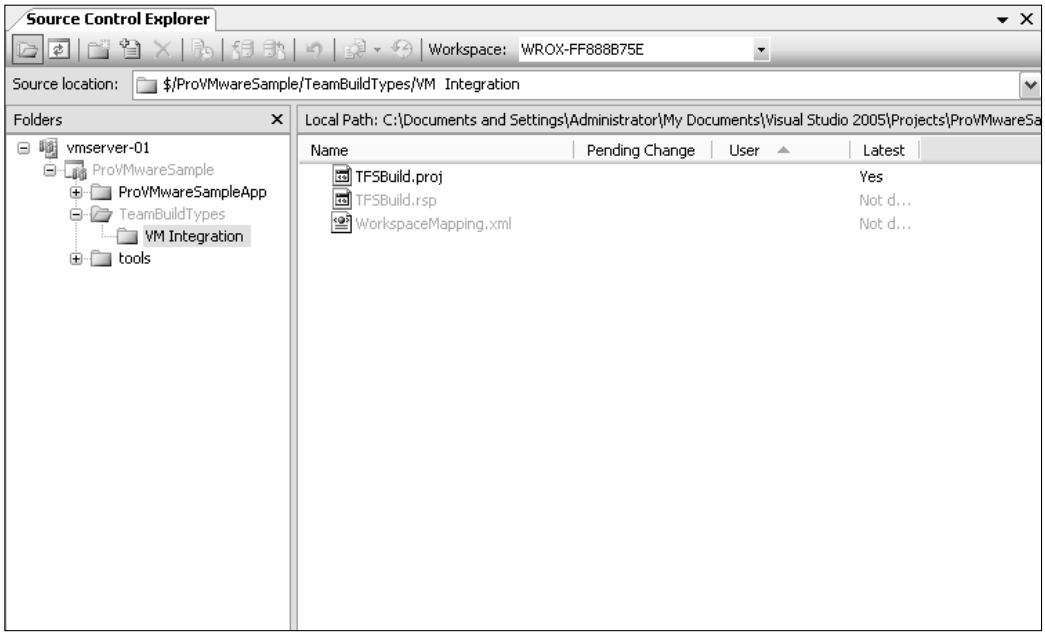

Figure 11-49

3. After the very last closing tag for Item Group, just prior to the closing project tag insert the following text:

```
<UsingTask TaskName="VMIntegration.VMwareServer"
 AssemblyFile="$(SolutionRoot)\tools\VMIntegration.dll" />

<Target Name="AfterDropBuild">
  <VMwareServer SourceDir="$(SolutionRoot)" ServerName="vmserver-01"
    ServerPort="902" UserName="administrator" Password="Pass@word!"
    NamePattern="test-winxppro-sp2.*"/>
</Target>
```

   You will, of course, need to customize the attributes under the target section to match your environment.

4. Once this is complete, close the project file and from the Pending Changes tab at the bottom of the IDE check it into Source Control.

If the Pending Changes tab is not visible along the bottom of your IDE select View➪Other Windows➪Pending Changes to reveal the pane in your IDE.

That concludes the setup. It's quite a bit more involved than CCNet was for sure. Because VSTS requires you to customize the build process with custom build assemblies, this solution is quite complex. For review you created a new team project and added a simple development project to it, created a new build type to kick off nightly, and then created a custom assembly and referenced it from the build process to handle the VMware Server aspect of the integration.

From the Team Explorer window double-click the VMIntegration build type and execute the build. If all goes well you should end up with a successful build, a virtual machine that's running and has a copy of the current binaries available on it for QA to review.

VMware Server Integration under Linux — A Brief Look

Before ending this chapter, I want to talk briefly about continuous integration with VMware Server under Linux.

As I mentioned earlier in this chapter, including Linux-based guest machine in your integration environment is as simple as the Windows-based ones. The trick, however, is continuous integration, like I demonstrated with CCNet and VSTS. There are a few continuous integration packages available for Linux. Most notable would be Continuum. Continuum is a Java-based continuous integration package so it will of course work on Windows- and Linux-based servers alike. The reason I bring it up is it is probably one of the more known packages for Linux-based continuous integration.

There are, of course, other packages available. Any of these will work to accomplish what you did in this chapter with CCNet and VSTS so long as they support calling scripts externally. Because Perl is widely available on Linux-based systems, the example used under the CCNet section would work very well on most Linux-based continuous integration packages as well. Remember, the main goal is to integrate the build with VMware Server. So long as your build package can leverage Perl, or another VMware API, then you're set. There is little difference between the Windows- and Linux-based setups beyond that.

Summary

In this chapter I covered two very intense topics, integration into your CruiseControl.NET and Visual Studio Team System build processes. The intent of the examples was to demonstrate to you how this integration can be achieved. You witnessed what I call the two extremes here. CCNet was incredibly easy and provided easy hooks into the process but lacks some of the robustness of VSTS. VSTS, though harder when it comes to hooking into the build process, may actually see more use because it's commercial and geared toward larger groups. Both are excellent products and worthy of a look if you haven't already checked them out.

In the next chapter, I take you through a real-world example from top to bottom and talk about how you can integrate VMware Server into the entire process.

12

A Real-World Scenario

For the final chapter I want to run through a real-world scenario, top to bottom. Aside from the daunting tasks of requirements and specifications, a full-blown application requires a great deal of preparation. The days of just sitting down and coding are behind us, at least for a project of any magnitude.

Over the next several sections I take you from defining your development and testing environment to the integration of virtual machines. The difference between this and the chapters before it is the goal of tighter integration into what should closely represent a normal project in your shop. This will not, however, be a high-level overview. I walk through this process step-by-step.

> *You are likely to see some scripts here that you have already been exposed to in previous chapters; however, some new twists and concepts are added to increase their worth.*

Defining Your Environment

A project just isn't a project without a target environment. I may say that partly in jest, but in reality every development project you undertake is targeted against a specific environment. Line of business applications are typically targeted toward the common desktop operating systems, whereas something like antivirus or antispam systems are usually targeted toward server-based operating systems. The target environment is determined for almost every application you develop. When I start a new project, I break down the environmental requirements into four sections:

- Target platform
- Supported platforms
- Tools
- Source and process control

Target Platform

I start right at the top with target platform. To me the target platform is simply the "runs best" or "designed for" type of platform. Not unlike the "designed for Windows XP" logos you find on software packaging, the target platform is just that, where you intend your application to run. Though this may not be a difficult decision, it is in fact an important one.

You may ask just what this has to do with VMware Server. Well, the simple fact is that development tends to go a lot smoother if applications are developed on their target platform. That is not a hard and fast rule, but the fact is that an application targeted for Windows XP Professional is better developed on Windows XP Professional. This tends to help uncover more platform-specific problems during the development cycle than development on something other than the target platform would. Such development isn't or can't always be the case, obviously, but a fact nonetheless. For this reason your target platform will also be your development platform.

The target platform for this real-world scenario is Windows XP Professional.

Supported Platforms

Next you identify the supported platforms. This bears distinction from the target platform in that it encompasses all the platforms in which you support the application. It's rare in today's world that you write an application that just runs, or is only supported, on a single platform, or operating system. For that reason defining the list of support platforms is important.

Where does this fit into the topic of VMware Server? Supported platforms in this context represent your test bed. Platform testing is one of the most important aspects in commercial software development. You can test the application on any platform you wish; however, the list that you support must be tested against. This list will become very important when you start creating your test images in a few minutes.

The supported platform for this real-world scenario is Windows Server 2003.

Tools

Tools, of course, are the things you make software with. It's only fitting that they be a part of your environment definition because they almost completely drive the problem. Take the .NET Framework, for example. You can write your C# code in Notepad, but who does that. You'll either use Visual Studio or one of the free IDE tools like SharpDevelop. Regardless, the decision to use the .NET Framework in and of itself pretty much decides the previous two steps for you based upon the framework's supported platforms.

Some questions to ask yourself when choosing the tool to develop with are scalability and manageability. Because in this scenario you will be creating development images for the team of developers working on this project, you have to decide how easy it will be to use the tool or tools of choice in that environment. Of course, most of the time this isn't an issue; however, if you are imposing standards and specifications that you need to enforce on your development team, it can quickly become one.

The tool of choice for this scenario is Visual Studio 2005.

64-Bit Platforms

Before I move on I want to address the topic of 64-bit platforms. Now that they have made their way into mainstream computing, at least somewhat, you may want to target or test against these platforms. Keep your VMware Server host machine in mind when you make this selection. In order for you to leverage a 64-bit guest machine in VMware Server, the host machine's processor must be an approved and supported 64-bit processor. Check the VMware web site at `www.vmware.com` for the latest supported hardware list.

Source and Process Control

Finally, you select source and process control. Source control is the tool you utilize to track and maintain your code base throughout the development process. I cannot stress how important source control is. Even a single developer can benefit from using it. A well-managed source control system can store, track, and archive your code base well after its release. Now, this may not even be a choice for you. You may already have a system in place to handle source control. If you do, that's great, because you really should have. However, in case you do not some things need to be considered. Right off is the issue of integration. If integration is your goal, then a choice here that fits into whatever piece of software you are attempting to integrate with is vital. Take, for instance, CruiseControl.NET (CCNet). CCNet has a very specific set of source control tools that it provides that it can communicate with natively. That list is large; however, it does exist. Some forethought is required before the selection is made.

The source control tool for this scenario is Visual SourceSafe 2005.

Process control is a whole other beast. Process control may mean different things to different people but in this context it simply refers to how you manage the development process. This can include things like bug tracking, developer tasks, milestones, build release management, and so on. At the very least you should consider some sort of bug management especially if you have a decent sized team of developers. The ability to track and maintain a history of bugs both past and present will make current and future development project easier. This allows your developers to learn from past mistakes, and help desk personnel to effectively troubleshoot problems with released software. Chances are you are not integrating this into your VMware Server environment; however, it is of course a design decision you make during this phase of development. Many different packages exist, too many to name; however, some provide APIs and software development kits that could allow you to integrate them into the process. You might do this to create a project milestone or a task that reflects the build and testing status. Because many source control providers allow you to assign labels you could easily reference the affected code base in such a task. This is of course a high-level discussion and beyond the scope of this book; however, it can be accomplished.

Defining Your Goals

How deep do you want to integrate? Ask yourself this question. Obviously the main goal of any development team is to write great software, but does an integrated environment help you accomplish that? Absolutely it can, but taken to the extreme it can actually slow you down.

The depth in which you integrate depends upon your workflow. If you find yourself shuffling around builds to testers, answering e-mails from QA about when and where they can test the application, or constantly slicking and reinstalling test banks and developer machines, you need some integration in your life. As I discussed many chapters ago, the virtual machine is simply the best way to write software. It doesn't fit every situation, but quite frankly the benefits outweigh the disadvantages. Sitting your developers down in front of a preconfigured, ready to run virtual machine is not only time efficient but makes for good consistent development environments. Your test banks are no different. Gone are the days where you had a binder full of Ghost images for every test machine and huge budget line items for the equipment. Now, with VMware Server, simply house multiple test machines on one server with non-persistent disks and not only will you save time with reimaging, but you'll save money in hardware. It's hard to argue against it.

The integration I am referring to doesn't just lie with the developer or test machines but with the entire process itself. As you have seen in the previous chapters, integrating continuous integration servers, like CruiseControl.NET or team development systems like Visual Studio Team System, makes life even easier. Going from the developer machine to source control to build management and testing ordinarily leaves you with many disconnected parts. Bridging those gaps with integration leaves you with a better overall process.

My goal with this scenario is to do just that, bridge the gaps. I have already decided that this project will use Visual Studio 2005, have a target of Windows XP Professional, and utilize Visual SourceSafe. Add to that VMware Server and build management as we have a rather slick setup.

Finally, I want to talk about overkill. Do you really need VMware Server for your project? That's a question you will have to answer yourself and for each project. It really boils down to the size of the project and the development workflow. The nice thing is that you already have it right there, so you might as well use it. The overkill here is complete integration between all the parts. VMware Server by itself is a wonderful tool, so are the others you use in your development, but the integration of all of them together can actually slow down a smaller project that doesn't need all of that. If you can develop, test, and ship within a week or two chances are most of your testing is done locally by your developers and the code base is rather small. This might not be the best choice for leveraging the integrated environment. That said, however, a large project spread out across several departments and several months is just screaming for some integration sweetness. Then you will see the increases in productivity and decreases in budgetary requirements.

Setting Up Your Development Images

All right, now it's time to get started. I've decided that this project will be written in .NET using Visual Studio 2005 and will target Windows XP Professional. My choice would be to house the developers' machines on Windows XP Professional as well. This will be a WinForms application so a straight standard install, fully updated, will work just fine. I want to walk through this step-by-step.

Now, because I Sysprep my images, I don't really care how many developers I have at the moment. All I know is that I need a developer image of Windows XP Pro, Visual Studio 2005, and Visual SourceSafe. Since you know how concerned I am about naming, how about I call them `dev-winxppro-sp2`. (I bet you saw that one coming!) I won't worry about affixing the project name to it at this time; however, remember that if you have multiple projects going at once you don't want your development images to

get mixed up. You should adopt a naming scheme that allows you to affix a project identifier as well. For my own projects I use a project management tool and assign each a three-digit identifier. That might seem a little quaint, but it works and can fit within the NETBIOS naming length. An example of that might be `dev124-xxp-sp2`. You'll notice I had to shorten the OS name, but it is still understandable. I always archive the entire development image after a big project, and this naming scheme helps me to identify what machines went with which projects. But because we aren't doing anything that elaborate here, the simple name will suffice.

Setting up the Virtual Development Machine

Fire up VMware Server (VMware Server v.1.0.0 in this case) on your server (Windows Server 2003 Standard with SP1) and get going. Figure 12-1 shows you where you are starting out. You want to create a new virtual machine with the previously mentioned operating system, 512MB of RAM each, no floppy drive, and bridged networking. (For this scenario, virtual machines will be stored in `C:\Virtual Machines`.)

> *If you already have a base image for Windows XP Professional, you can use it instead of creating a new one. This will cut down on your setup time. Refer to Chapter 4 for a discussion of base images and their use with VMware Server.*

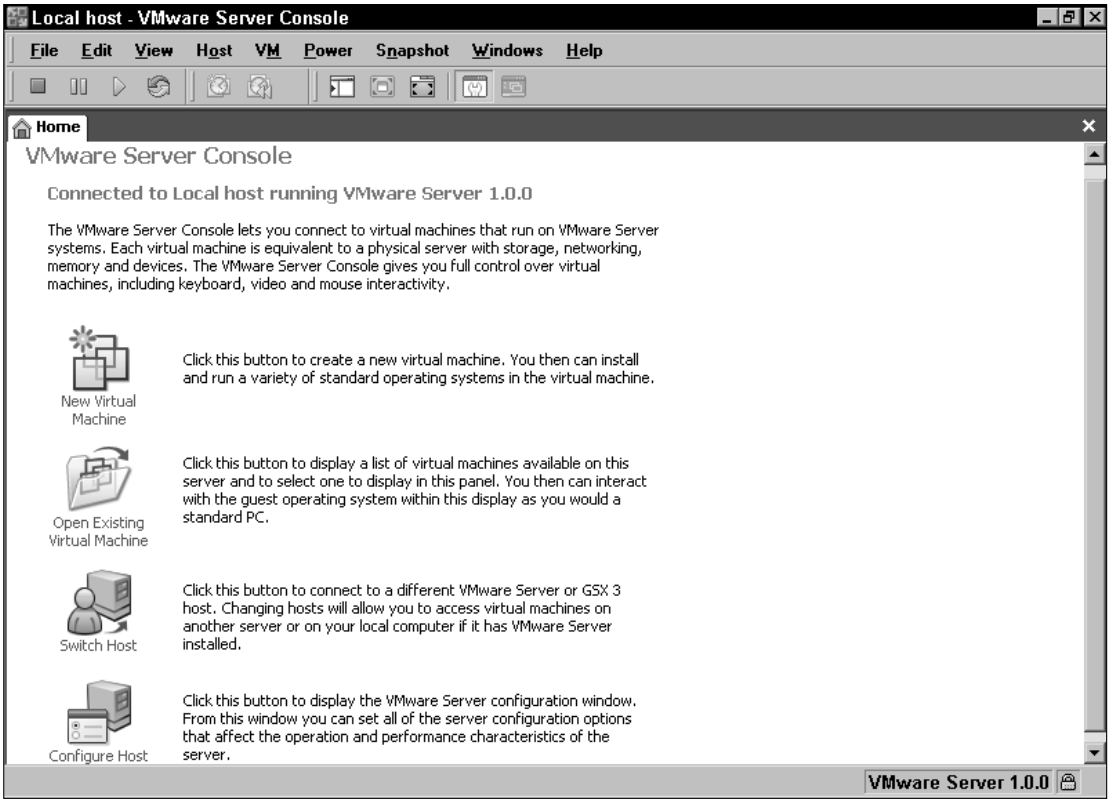
Local host - VMware Server Console
File Edit View Host VM Power Snapshot Windows Help
Home
VMware Server Console
Connected to Local host running VMware Server 1.0.0
The VMware Server Console lets you connect to virtual machines that run on VMware Server systems. Each virtual machine is equivalent to a physical server with storage, networking, memory and devices. The VMware Server Console gives you full control over virtual machines, including keyboard, video and mouse interactivity.
New Virtual Machine
Click this button to create a new virtual machine. You then can install and run a variety of standard operating systems in the virtual machine.
Open Existing Virtual Machine
Click this button to display a list of virtual machines available on this server and to select one to display in this panel. You then can interact with the guest operating system within this display as you would a standard PC.
Switch Host
Click this button to connect to a different VMware Server or GSX 3 host. Changing hosts will allow you to access virtual machines on another server or on your local computer if it has VMware Server installed.
Configure Host
Click this button to display the VMware Server configuration window. From this window you can set all of the server configuration options that affect the operation and performance characteristics of the server.
VMware Server 1.0.0

Figure 12-1

Remember, I said I would take you through this step-by–step, so here you go.

1. Click New Virtual Machine from the Home tab within the VMware Server Console. This will bring up the now-familiar New Virtual Machine Wizard, as shown in Figure 12-2. You're going to do a custom setup here, so there are going to be quite a few steps to this.

New Virtual Machine Wizard
Welcome to the New Virtual Machine Wizard
This wizard will guide you through the steps of creating a new virtual machine.
VMware Server
CONSOLE
< Back
Next >
Cancel

Figure 12-2

2. Because you want to set up as much as possible through the wizard, click Next and select Custom from the choices in Figure 12-3. This will allow you a great deal more flexibility with your virtual machine configuration. Click Next.

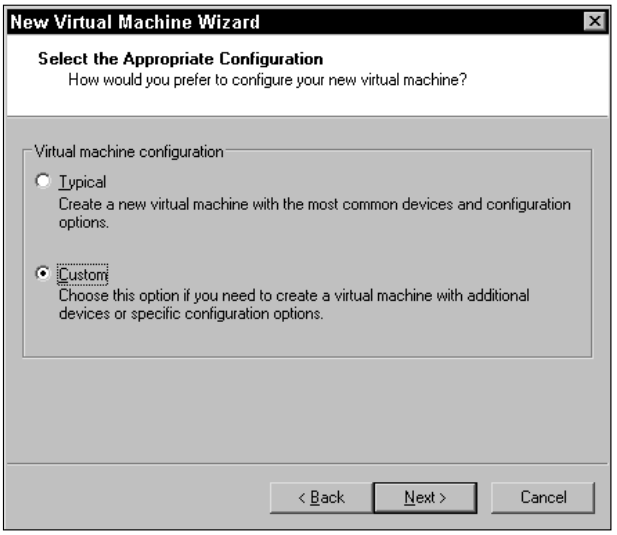
New Virtual Machine Wizard
Select the Appropriate Configuration
How would you prefer to configure your new virtual machine?
Virtual machine configuration
Typical
Create a new virtual machine with the most common devices and configuration options.
Custom
Choose this option if you need to create a virtual machine with additional devices or specific configuration options.
< Back
Next >
Cancel

Figure 12-3

There will, however, still be a few manual steps that I recommend you take after the wizard just to help things go a little smoother. I discuss those a bit later in the chapter after you've completed your work in the wizard.

3. The first real step in the custom wizard setup is, of course, the base operating system type. Because this development image is Windows XP Professional, select Microsoft Windows and then choose Windows XP Professional from the drop-down list shown in Figure 12-4. Click Next.

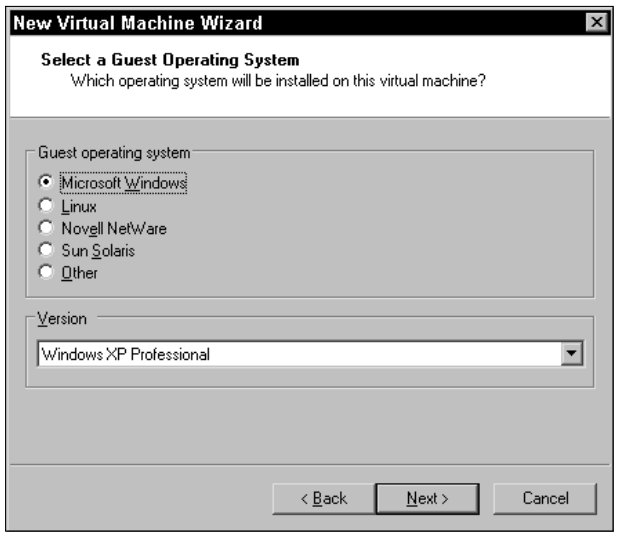

Figure 12-4

4. Next you enter the name of your virtual machine. As I pointed out previously you want to call this one `dev-winxppro-sp2`. Enter this name, or a name that fits into your scheme, in the Virtual Machine Name text box. Make sure the directory name is correct below it as well. The wizard should update the path to reflect the name you entered above it. It's no problem if you have to move it later, but you might as well get it right first. Figure 12-5 illustrates the name and path entered for this example. Click Next.
5. Now you move on to access rights. By default all new virtual machines are created private. If you remember from Chapter 3, I discussed the various ways to restrict access to virtual machines. It basically boils down to file system permissions. For now, leave the Make this virtual machine private option, shown in Figure 12-6, selected and move on by clicking Next.
6. The Startup/Shutdown options are important:
   - If you leave the default selected, User that powers on the virtual machine, then you will be unable to close the console window you start the machine from. It basically just runs in the current instance under the current user's security context.
   - Now, I typically choose Local system account here because it's easy and I don't have to worry about security permissions with physical hardware usage on the server by the

virtual machine. If you remember from before certain pieces of physical hardware, like SCSI controllers, require the user that is running the virtual machine to have certain security rights on the server. Using the local system account takes care of that.

- If you so choose you can also specify what user you want it to run under.

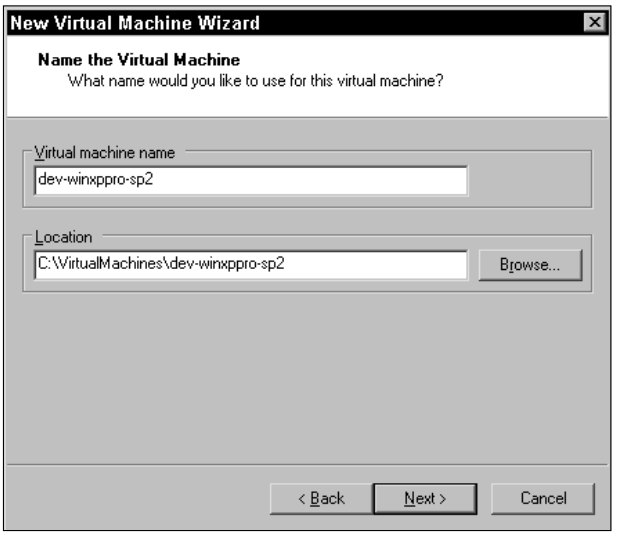

Figure 12-5

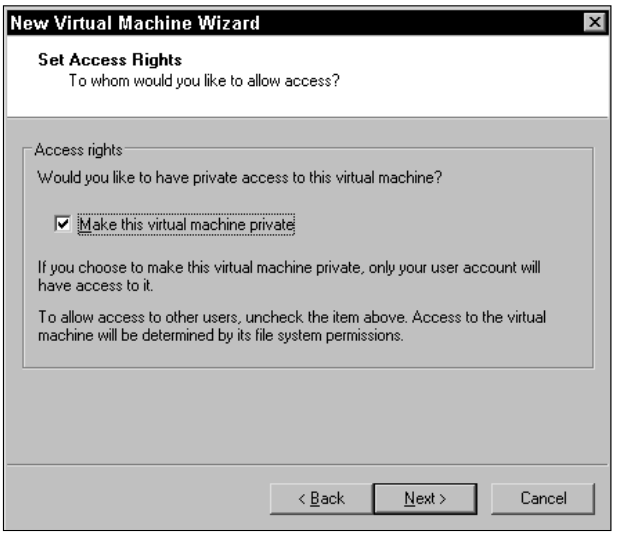
New Virtual Machine Wizard
Set Access Rights
To whom would you like to allow access?
Access rights
Would you like to have private access to this virtual machine?
Make this virtual machine private
If you choose to make this virtual machine private, only your user account will have access to it.
To allow access to other users, uncheck the item above. Access to the virtual machine will be determined by its file system permissions.
< Back
Next >
Cancel

Figure 12-6

As I indicated, local system is my normal choice, so I'm going to have you stick to that here as well. Select the second option shown in Figure 12-7 and click Next. For now, at least, leave the power on and off options at the bottom of this wizard step alone.

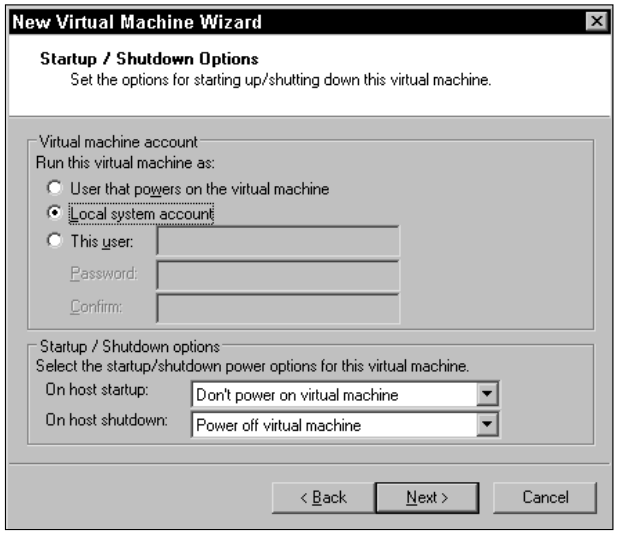
New Virtual Machine Wizard
Startup / Shutdown Options
Set the options for starting up/shutting down this virtual machine.
Virtual machine account
Run this virtual machine as:
User that powers on the virtual machine
Local system account
This user:
Password:
Confirm:
Startup / Shutdown options
Select the startup/shutdown power options for this virtual machine.
On host startup:
Don't power on virtual machine
On host shutdown:
Power off virtual machine
< Back
Next >
Cancel

Figure 12-7

7. Now you need to select the number of processors you want to allocate to your virtual machine. Don't be caught off guard by this wizard step. If your server has only one processor this step will still allow you to select two. At this time VMware Server will only allow you to configure up to two physical processors to share with the virtual machine. The virtual SMP support is still somewhat spotty, so I recommend you stick with just a single processor as shown in Figure 12-8. Click Next.

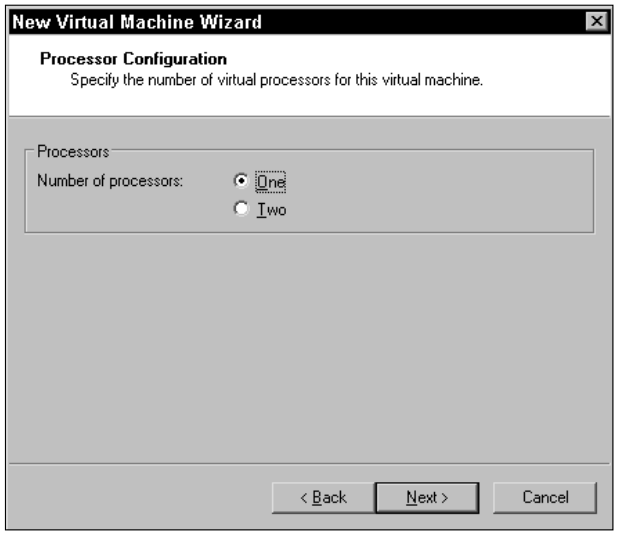

Figure 12-8

8. The next step concerns the memory for your virtual machine. During the introduction to this section, I stated allocating 512MB for the virtual machine. You can either type in this number in the box to the right of the slider or try your luck at hitting 512 exactly using the slider control. As far as the recommended settings at the bottom of this screen, I wouldn't pay much attention to them other than the maximum available. You'll notice that I have only 796MB available for any given virtual machine. I should point out that my test machine has only 1GB total. The rest of the memory is deliberately sectioned off my VMware Server to allow for host system operation. I highly suggest you not crank up a single machine to the maximum allowed. Figure 12-9 shows you where you are at for this example. Click Next.

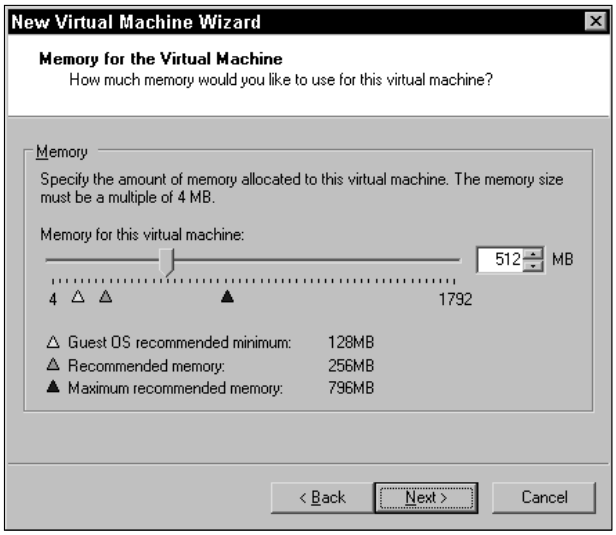
New Virtual Machine Wizard
Memory for the Virtual Machine
How much memory would you like to use for this virtual machine?
Memory
Specify the amount of memory allocated to this virtual machine. The memory size must be a multiple of 4 MB.
Memory for this virtual machine:
512
MB
4
1792
Guest OS recommended minimum: 128MB
Recommended memory: 256MB
Maximum recommended memory: 796MB
< Back
Next >
Cancel

Figure 12-9

9. Next, you deal with networking for your virtual machine. If you need some clarification on the networking options, I encourage you to jump back into Chapter 3 where I went over the different virtual network types. For this example I want the virtual machine to be on the same network as the rest of my environment, so I'm going to select bridged networking from the choices available. Figure 12-10 provides you with the complete list and the selection I made. Press Next when you are satisfied with your choice.

10. Next you move on to adapter types. Because this is Windows XP, the wizard will configure you with an IDE adapter, so the choice of SCSI adapters here is irrelevant. I'm going to leave the default SCSI choice for now and continue on. Figure 12-11 will catch you up. Click Next.

11. Selecting your virtual disk is the next step. Here you choose to create a new virtual disk, use an existing one, or use a physical disk. For this example you need to create a new virtual disk; however, it is well within the realm of possibility that you may have an existing disk to use. The third option, Use a physical disk, is a rather complicated one. The VMware Server documentation can go into that one further. For now select new virtual disk, as shown in Figure 12-12, and click Next.

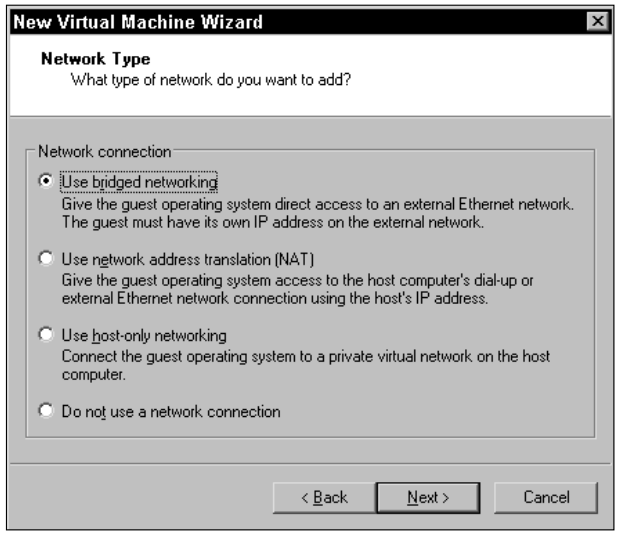
New Virtual Machine Wizard
Network Type
What type of network do you want to add?
Network connection
Use bridged networking
Give the guest operating system direct access to an external Ethernet network. The guest must have its own IP address on the external network.
Use network address translation (NAT)
Give the guest operating system access to the host computer's dial-up or external Ethernet network connection using the host's IP address.
Use host-only networking
Connect the guest operating system to a private virtual network on the host computer.
Do not use a network connection
< Back
Next >
Cancel

Figure 12-10

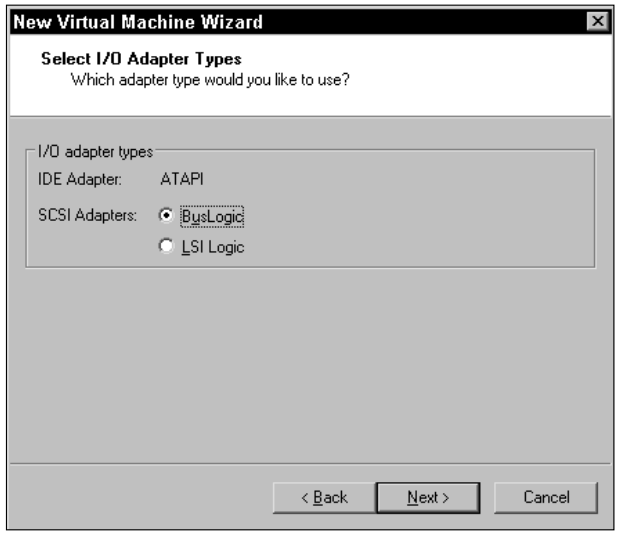

Figure 12-11

The most common use for a physical disk setup is taking an existing system set up for multiboot into different operating systems and running one of them within a virtual machine. This is not as simple as it may sound, however, because there are a number of conditions to account for in order for it to work. Visit the VMware forums at `www.vmware.com/community` *and the VMware Server documentation for more information.*

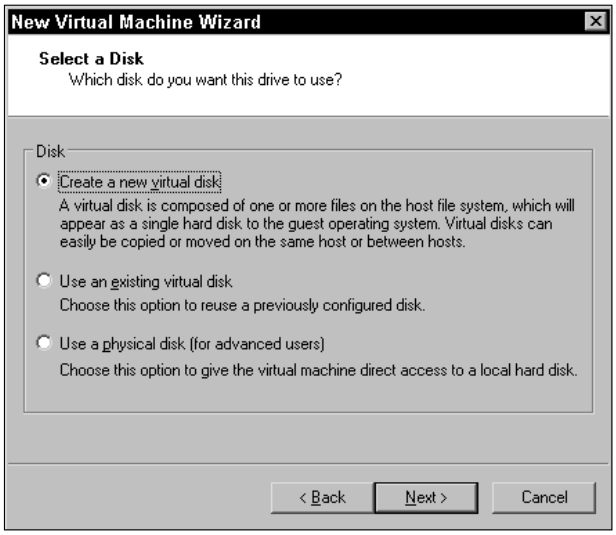
New Virtual Machine Wizard
Select a Disk
Which disk do you want this drive to use?
Disk
Create a new virtual disk
A virtual disk is composed of one or more files on the host file system, which will appear as a single hard disk to the guest operating system. Virtual disks can easily be copied or moved on the same host or between hosts.
Use an existing virtual disk
Choose this option to reuse a previously configured disk.
Use a physical disk (for advanced users)
Choose this option to give the virtual machine direct access to a local hard disk.
< Back
Next >
Cancel

Figure 12-12

12. Only three wizard steps to go. In the next screen you select the type of adapter to use for the virtual disk. For Windows XP you will get better performance from the IDE adapter, so their recommended choice is a good one. Leave the default of IDE shown in Figure 12-13 selected and click Next.

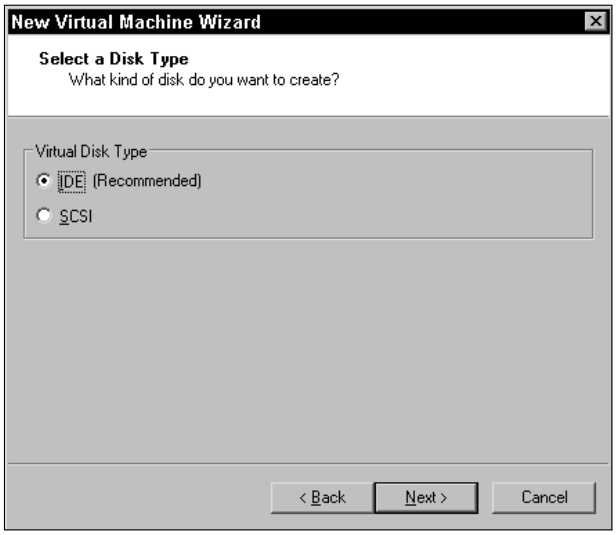

Figure 12-13

13. Ah, the size of the virtual disk. You need to think carefully here. I can tell you from experience that Windows XP Professional with SP2 plus Visual Studio 2005 fills over 10GB worth of disk space. For this specific virtual machine I recommend somewhere between 15 and 20GB. To be on the safe side, I suggest selecting 20GB.

Now, as you can see from Figure 12-14, you have a couple more options to peruse:

- The Allocate all disk space now selection is all about performance. If you deselect this, the virtual machine will be created much faster; however, you could experience performance hits as the virtual disk expands over time. If you have the disk space I recommend you allocate everything now. Just keep in mind that when it comes time to archive this image you will have to deal with that 20GB pre-allocated disk instead of one that is possibly smaller.
- Lastly is the option to split into 2GB files. To be completely honest with you I never use this; however, I could see some advantages to it — most notably the ability to split the disks across media if you were to archive to DVD or something similar.

Select 20GB for the virtual disk size and allocate it all now.

New Virtual Machine Wizard
Specify Disk Capacity
How large do you want this disk to be?
Disk capacity
This virtual disk can never be larger than the maximum capacity that you set here.
Disk size (GB):
20.0
Allocate all disk space now.
By allocating the full capacity of the virtual disk, you enhance performance of your virtual machine. However, the disk will take longer to create and there must be enough space on the host's physical disk.
If you do not allocate disk space now, your virtual disk files will start small, then become larger as you add applications, files, and data to your virtual machine.
Split disk into 2 GB files
< Back
Next >
Cancel

Figure 12-14

14. Final step. Here the wizard asks if you want to name your virtual disk file something other than the default. I recommend you match this with the name of the virtual machine itself. It will save you later if things get separated out. Change the name to `dev-winxppro-sp2.vmdk`, as shown in Figure 12-15.

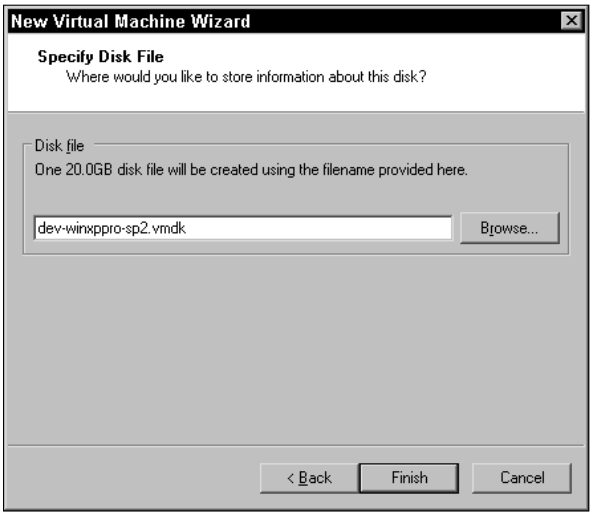

Figure 12-15

Click Finish and prepare for a wait. Pre-allocating the virtual disk takes time, so grab some coffee. Once it is complete you should have a brand new virtual machine in your server console as shown in Figure 12-16. You now have a few manual steps that need to be done.

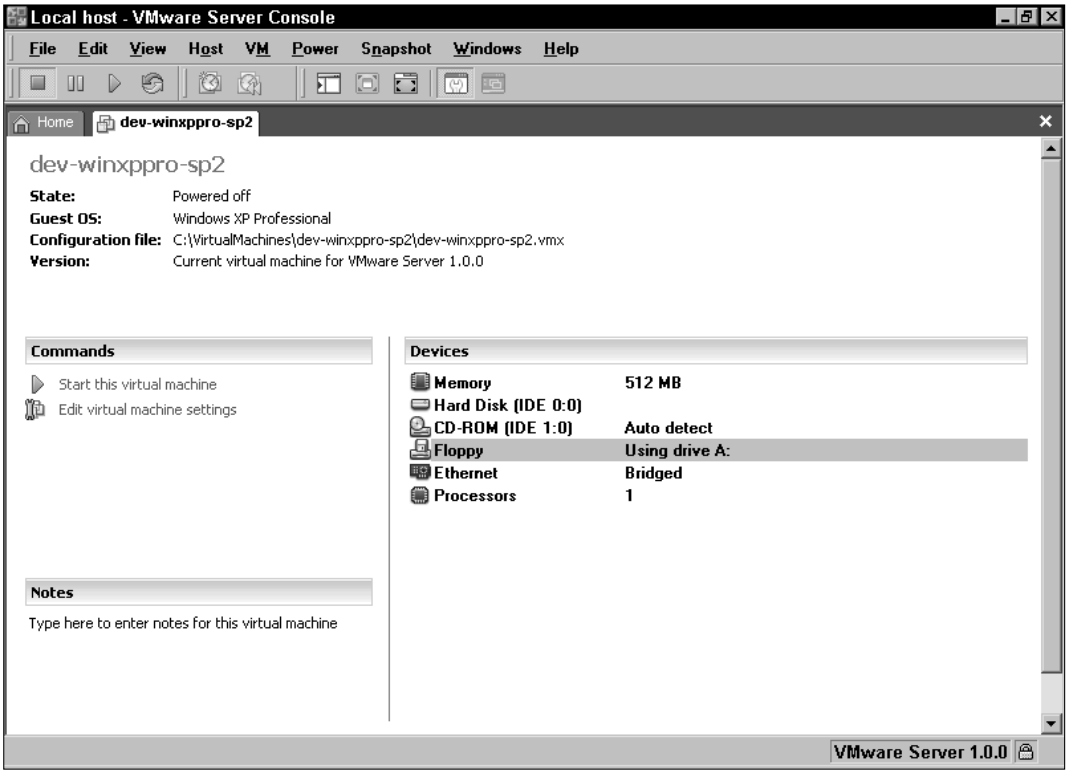

Figure 12-16

Deleting the Virtual Development Machine Floppy Drive

First and foremost, you need to get rid of that floppy drive. I'm not sure why VMware still defaults new virtual machines to have a floppy drive but if you don't need it, and chances are you don't, get rid of it. The number one cause of virtual machines entering a stuck state on startup is due to the floppy drive. Leave it in there and you'll see what I mean. Click Edit virtual machine settings in the left-hand pane to reveal the screen in Figure 12-17.

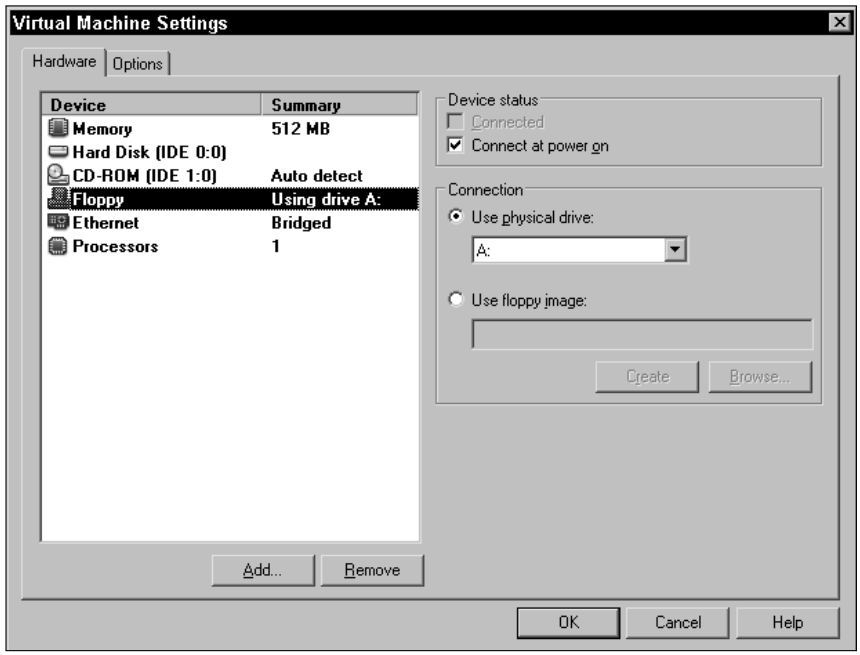

Figure 12-17

Highlight the floppy drive and click the Remove button in the lower part of the left-hand pane. Once it has been removed, click OK to return to the server console.

Renaming the Virtual Development Machine Configuration File

The last manual step is to rename that virtual machine configuration file. By default the configuration file is named after the operating system that it was set up for. As you already know from previous chapters this doesn't help you much when parsing via code through lists of virtual machines available on the server. Start by right-clicking the virtual machine's tab in the server console and selecting Remove from Inventory, as shown in Figure 12-18.

Once you have removed the machine from inventory open up My Computer and browse to the location where you store your virtual machines. Mine are located in `C:\VirtualMachines`. This directory, if it's where you store all your virtual machines, could contain several folders. You are, of course, looking for `dev-winxppro-sp2`. Double-click it to reveal Figure 12-19.

Figure 12-18

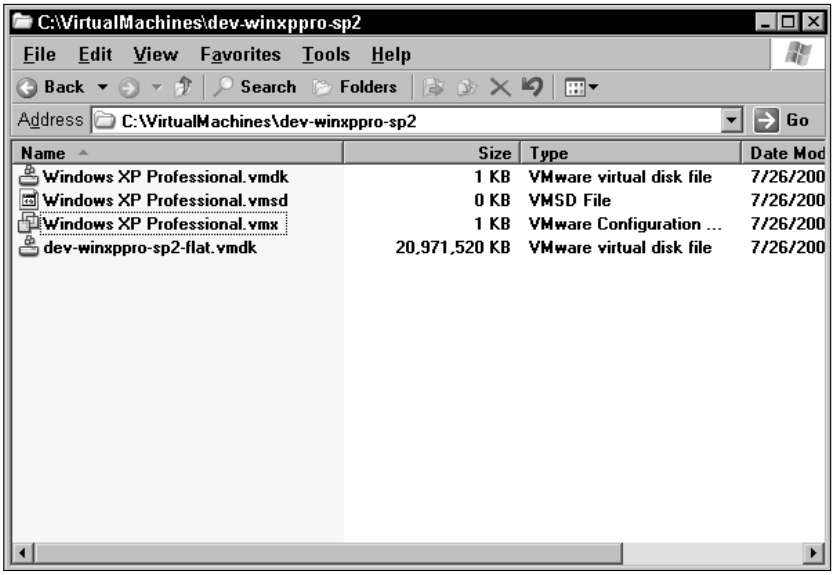
C:\VirtualMachines\dev-winxppro-sp2
File Edit View Favorites Tools Help
Back Search Folders
Address C:\VirtualMachines\dev-winxppro-sp2
Go
Name Size Type Date Mod
Windows XP Professional.vmdk 1 KB VMware virtual disk file 7/26/200
Windows XP Professional.vmsd 0 KB VMSD File 7/26/200
Windows XP Professional.vmx 1 KB VMware Configuration ... 7/26/200
dev-winxppro-sp2-flat.vmdk 20,971,520 KB VMware virtual disk file 7/26/200

Figure 12-19

The file of interest here is the configuration file named `Windows XP Professional.vmx`. It seems to me a good choice here for VMware would be to name the configuration file the same as the virtual

machine's name. If you don't make this change, any code you use to parse through the inventory would reveal a list of possibly dozens of configuration files with the same name. Of course you could always split out the directory name, if you set that to be different; however, it still seems like it would be nice for this to be done automatically.

Right-click the configuration file and rename it to `dev-winxppro-sp2.vmx`. Once this is complete you can close the My Computer window and revert back to the server console.

Because you removed this machine from inventory to rename its configuration file, you need to add it back. From the Home tab select the Open Existing Virtual Machine icon to reveal the dialog in Figure 12-20.

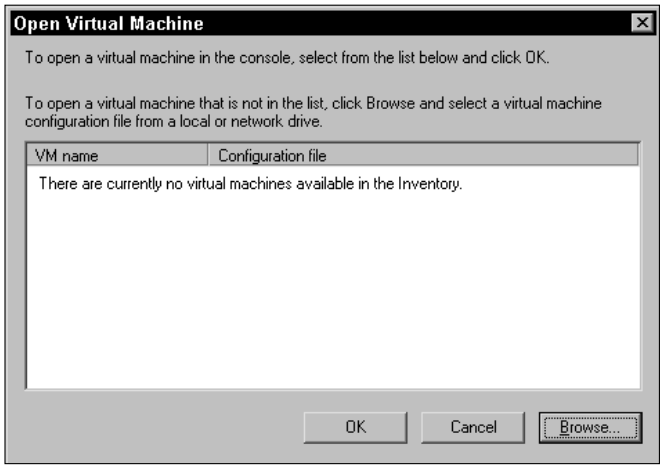
Open Virtual Machine
To open a virtual machine in the console, select from the list below and click OK.
To open a virtual machine that is not in the list, click Browse and select a virtual machine configuration file from a local or network drive.
VM name
Configuration file
There are currently no virtual machines available in the Inventory.
OK
Cancel
Browse...

Figure 12-20

Select Browse and navigate to the folder where you stored your virtual machine. Once there double-click its configuration file to add it back into inventory.

Setting up the Virtual Development Machine Software for Deployment

Well, the development virtual machine is configured and ready for software. Slap in your Windows XP installation media and start installing. Remember, you will be running Sysprep on this image in a moment, so make sure you install everything you want on the image now and set up any configuration items in your applications for your developers. This is the stage where you can enforce some specifics on your developers right out of the box. Also, don't forget to install the VMware Tools package on the guest machine. Once everything is installed and you've made sure it works correctly, it's time to Sysprep it.

If you don't remember Sysprep, you can refer back to the freezing your guest section in Chapter 4. Sysprep, or the System Preparation Tool, is used to roll the operating system back into a mini-setup state. This allows for a few benefits. The most important in this scenario is the ability to give each machine a name and security identifier. This is vital if you want them to co-exist on a network.

If you remember from Chapter 4 a different version of Sysprep exists for every Windows operating system and service pack level. You must be sure you are using the proper version of Sysprep before moving on. For this scenario you are using Windows XP Professional with SP2 so there are two places you can locate the proper Sysprep files for it:

- First, the `Support/Tools` folder located on the original Windows XP Professional SP2 installation media. You are looking for the `deploy.cab` file located in that folder. If this is your chosen method copy this file to the desktop of your guest machine now. You will do more with it in a minute.
- The second method is to download the Sysprep files for your specific operating system and build from the Microsoft web site. Simply go to the main Microsoft web site and search for your operating system name, service pack level, and the text *Deployment Tools*. An example might be a search for *Windows XP SP2 Deployment Tools*. This should lead you to the proper download.

No matter which way you choose to get the deployment tools you should now have a `.cab` file on the desktop of your guest machine. Double-click this file to reveal its contents. You should see something similar to Figure 12-21.

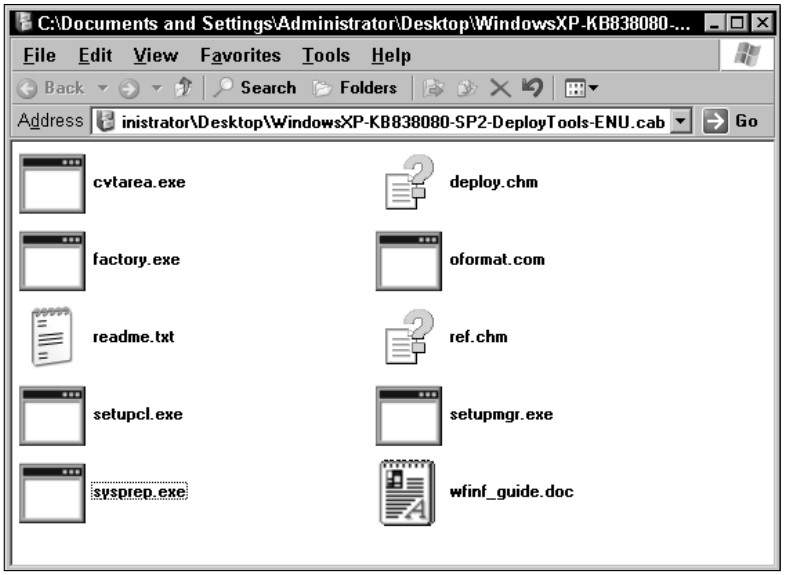

Figure 12-21

Now, if your memory serves you, you'll recall that Sysprep likes to run from within a folder under the root named `sysprep`. Open My Computer and go into your C: drive. From there create a new folder name `sysprep`. Now, jump back to the `.cab` file with the distribution tools and copy out just three of the files. That's all you are interested in. Grab `setupcl.exe`, `setupmgr.exe`, and `sysprep.exe`. Copy these files into the `C:\sysprep` folder you just created. Once you are done it should look just like Figure 12-22.

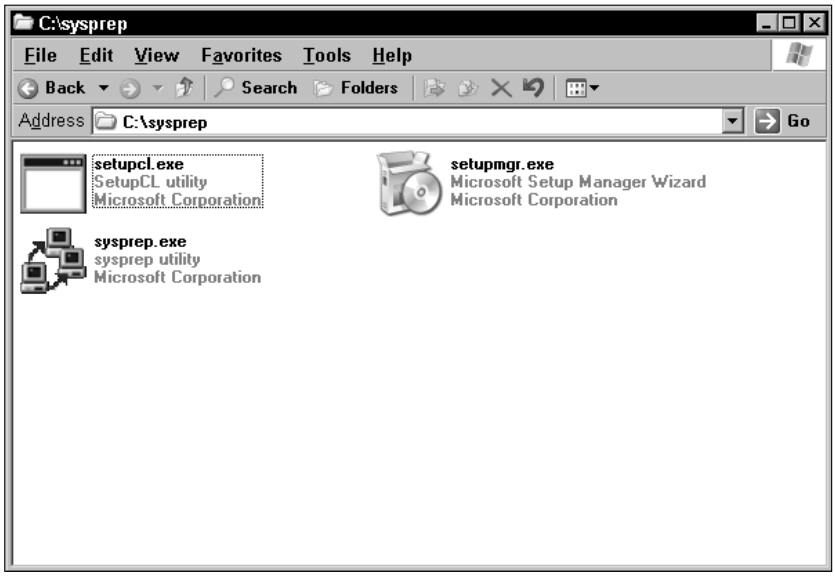

Figure 12-22

The first thing you have to do is create an answer file:

1. You can do this with the `setupmgr.exe` application. Go ahead and double-click it now. You will be greeted with a standard welcome page; click Next to reveal Figure 12-23, your starting point.

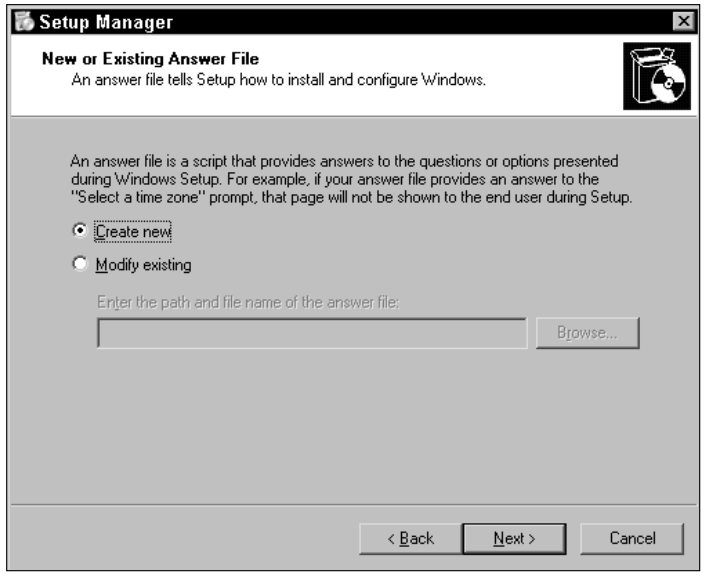
Setup Manager
New or Existing Answer File
An answer file tells Setup how to install and configure Windows.
An answer file is a script that provides answers to the questions or options presented during Windows Setup. For example, if your answer file provides an answer to the "Select a time zone" prompt, that page will not be shown to the end user during Setup.
Create new
Modify existing
Enter the path and file name of the answer file:
Browse...
< Back
Next >
Cancel

Figure 12-23

2. For this example, you are going to create a new answer file; however, if you still have the answer file you created in Chapter 4, you can apply it here by using the Modify existing selection and browsing to the file. Select your choice and click Next to continue to Figure 12-24, the setup type screen.

3. The Sysprep type is not selected by default so be sure you select it and click Next. I'm going to skip the next screenshot because it is simply selecting your operating system. It is, of course, Windows XP Professional. Select it and click Next. You will land on Figure 12-25.

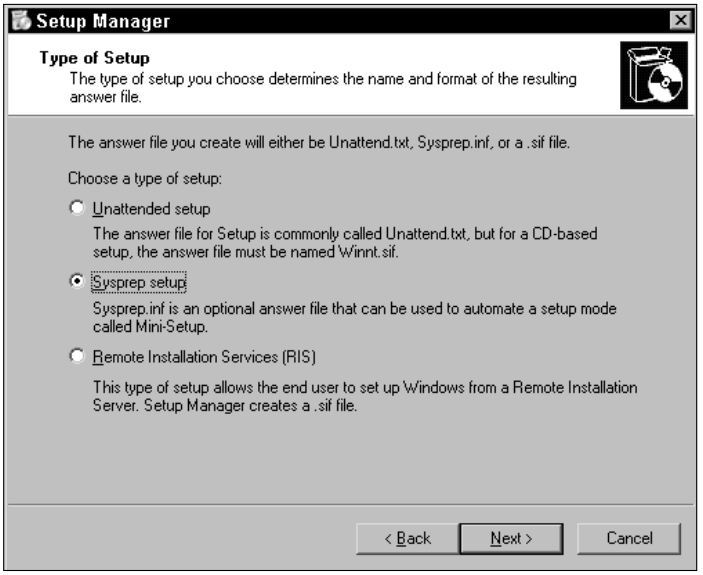
Setup Manager
Type of Setup
The type of setup you choose determines the name and format of the resulting answer file.
The answer file you create will either be Unattend.txt, Sysprep.inf, or a .sif file.
Choose a type of setup:
Unattended setup
The answer file for Setup is commonly called Unattend.txt, but for a CD-based setup, the answer file must be named Winnt.sif.
Sysprep setup
Sysprep.inf is an optional answer file that can be used to automate a setup mode called Mini-Setup.
Remote Installation Services (RIS)
This type of setup allows the end user to set up Windows from a Remote Installation Server. Setup Manager creates a .sif file.
< Back
Next >
Cancel

Figure 12-24

Setup Manager
License Agreement
Do you accept the terms of the License Agreement for Windows?
Using Sysprep, you can fully automate a Windows installation so that no user input is required.
To use this option, you must accept the terms of the End User License Agreement (EULA) and any Microsoft license agreements you have for the version of Windows you want to install. For more information about the EULA, consult your documentation or your Microsoft license agreement.
Do you want to fully automate the installation?
Yes, fully automate the installation
No, do not fully automate the installation
If you choose No, the end user must accept the End User License Agreement.
< Back
Next >
Cancel

Figure 12-25

4. This step is important. If you choose to fully automate the installation, you will not be prompted for any information during the mini-setup. Because you not only want to generate a unique SID but assign a new machine name, you need the No selection here. Click Next.

5. That takes youth the *Setup Manager* screen shown in Figure 12-26. This is the meat and potatoes of the mini-setup.

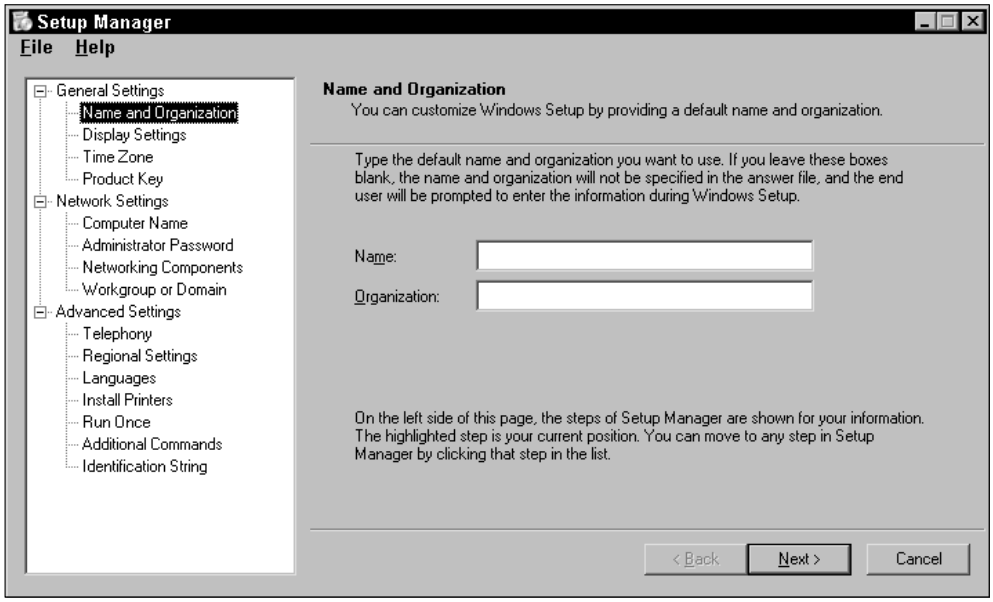
Setup Manager
File Help
General Settings
Name and Organization
Display Settings
Time Zone
Product Key
Network Settings
Computer Name
Administrator Password
Networking Components
Workgroup or Domain
Advanced Settings
Telephony
Regional Settings
Languages
Install Printers
Run Once
Additional Commands
Identification String
Name and Organization
You can customize Windows Setup by providing a default name and organization.
Type the default name and organization you want to use. If you leave these boxes blank, the name and organization will not be specified in the answer file, and the end user will be prompted to enter the information during Windows Setup.
Name:
Organization:
On the left side of this page, the steps of Setup Manager are shown for your information. The highlighted step is your current position. You can move to any step in Setup Manager by clicking that step in the list.
< Back
Next >
Cancel

Figure 12-26

Now, Sysprep doesn't like blank fields. Anything here that you leave blank the mini-setup program will ask you for. I encourage you to fill in as much as possible that will be true for all the development machines. The most important field to leave blank is the computer name under the Network Settings column. This should be set to manual and left blank so the mini-setup will know to ask you for the information. The rest of the information under Advanced Settings can be left alone; it will not prompt you for any of that information.

6. To finish up the Setup Manager, click the Identification String node and then click the Finish button in the lower right. The wizard will prompt you for a location to save the answer file to, as shown in Figure 12-27. If not already directed into the `C:\sysprep` folder, make sure the file is saved there. Click OK.

7. I want to bring special attention to this next screen shown in Figure 12-28. After you click OK on the preceding dialog you will be faced with a Completing Setup Manager screen where your only choice is the Cancel button. Don't let this screen fool you. The answer file is already saved, just go ahead and click Cancel; nothing bad will happen.

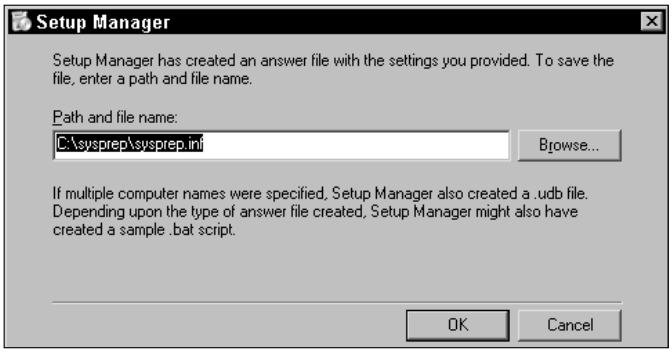
Setup Manager
Setup Manager has created an answer file with the settings you provided. To save the file, enter a path and file name.
Path and file name:
C:\sysprep\sysprep.inf
Browse...
If multiple computer names were specified, Setup Manager also created a .udb file. Depending upon the type of answer file created, Setup Manager might also have created a sample .bat script.
OK
Cancel

Figure 12-27

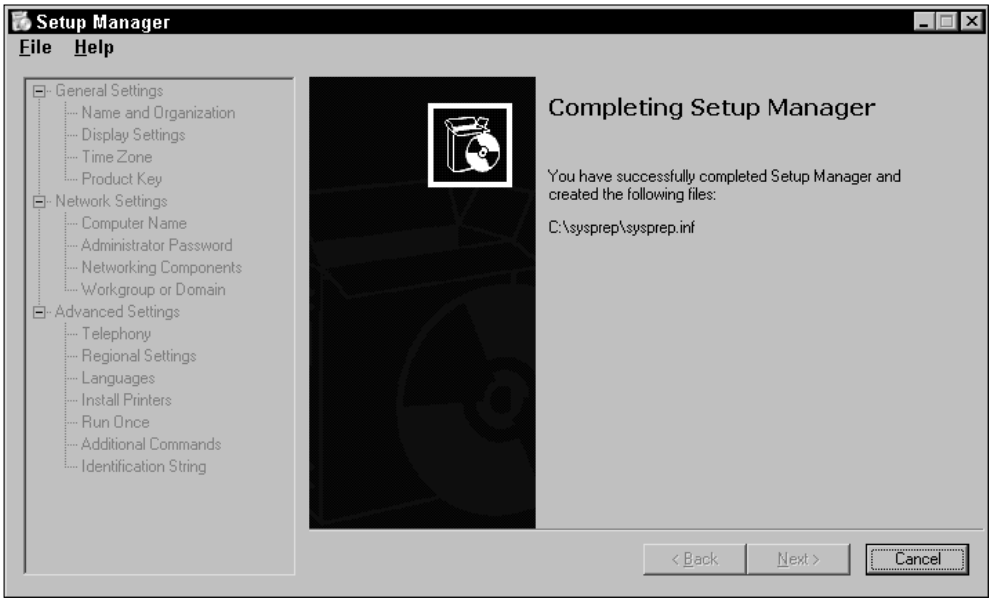

Figure 12-28

Now that you have your answer file, you should have two new items in your `C:\sysprep` folder. I've attached a shot of mine in Figure 12-29 for reference. The newly created answer file is there (`sysprep.inf`) as well as a new folder. Don't worry about the folder; it's most likely empty but is required by Sysprep nonetheless.

> *A word of caution here. Once you package up this installation, on the next startup Sysprep will delete the entire* `C:\sysprep` *folder as part of its routine. I recommend you zip up this folder and either copy it to your host machine or somewhere on the guest. If the Sysprep answer file misbehaves, you won't have to go through the process of pulling the files out of the .cab and creating a new answer file again. Just extract the archive and reconfigure the answer file using the Use existing option and loading the existing answer file.*

All right, you've archived the Sysprep folder, so now it's time to package it up. Go ahead and double-click the `sysprep.exe` file located in the `C:\sysprep` folder. You should see the main Sysprep tool screen, shown in Figure 12-30.

At the conclusion of the next step Sysprep will shut down the guest machine. Make sure you are ready for this to happen before you continue on.

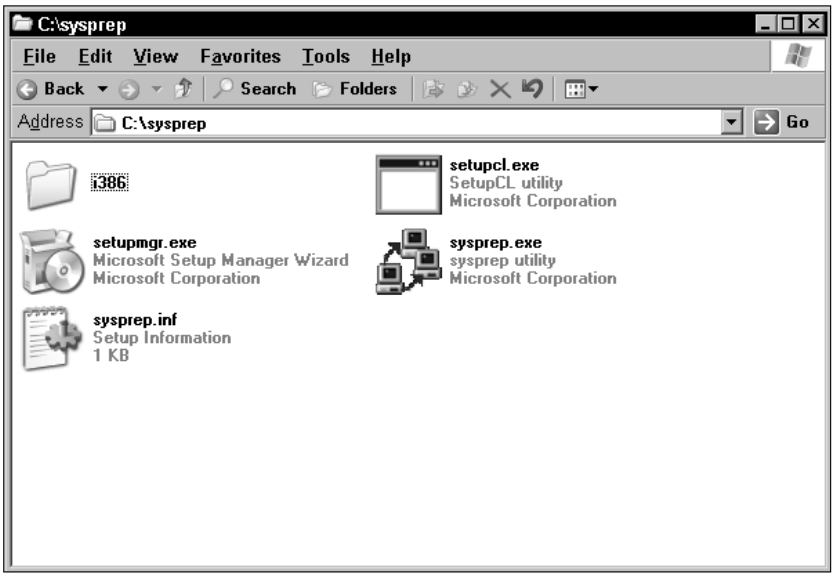

Figure 12-29

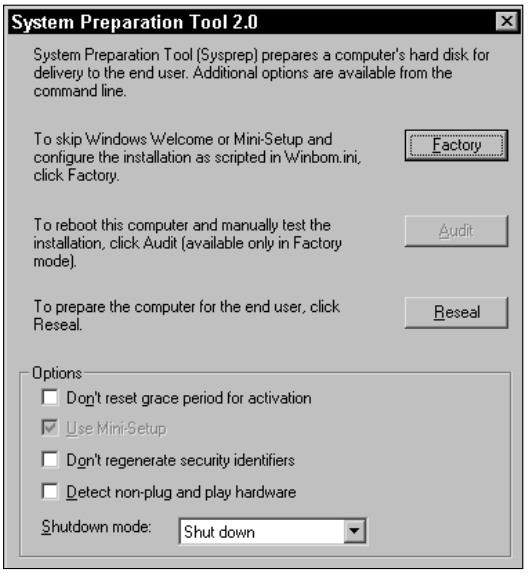
System Preparation Tool 2.0
System Preparation Tool (Sysprep) prepares a computer's hard disk for delivery to the end user. Additional options are available from the command line.
To skip Windows Welcome or Mini-Setup and configure the installation as scripted in Winbom.ini, click Factory.
Factory
To reboot this computer and manually test the installation, click Audit (available only in Factory mode).
Audit
To prepare the computer for the end user, click Reseal.
Reseal
Options
Don't reset grace period for activation
Use Mini-Setup
Don't regenerate security identifiers
Detect non-plug and play hardware
Shutdown mode:
Shut down

Figure 12-30

There are, as you can tell, several options here. The only one you are interested in is Reseal. Make sure the Use Mini-Setup box is selected and click the Reseal button. After a few moments the guest machine will shut down.

Depending upon the guest operating system and version of Sysprep used the Mini-Setup checkbox may or may not be grayed out. Regardless make sure it is selected. If it is grayed out, it should be selected by default. If not, review your Sysprep answer file, ensure you have the right version of Sysprep for the operating system in use, and finally refer to the Sysprep documentation if you are still having trouble.

Now, and I can't stress this enough, the most important thing you can do at this point is start the machine right back up. I recommend this because you need to make sure that Sysprep and the answer file you created perform as you expected. Do this now instead of realizing it later.

Once you start the machine back up, if the mini-setup performs to your expectations, copy your archived version of the Sysprep folder back onto the guest machine's C: drive and repeat the Reseal procedure. Once the machine shuts down you can consider your development image complete.

You can now copy and distribute this image to your development team. Remember that the virtual machines you create with VMware Server will work with VMware Workstation. If your environment calls for the developers to work off a local virtual machine, then by all means use VMware Workstation and copy the images locally.

Setting Up Your Testing Images

The testing images are an important part of this scenario. Because your ultimate goal is to integrate VMware Server into your process as you did in the previous chapter, take a few minutes to go over your goals for this project in terms of testing.

While the target platform for this project is Windows XP Profession SP2, you also want to support Windows Server 2003. Because your platforms are twofold, you need at least two different testing machines with the operating systems just mentioned.

Because naming is always an issue, go with the scheme that follows:

- `test-xppro-sp2`
- `test-svr2k3-sp1`

Normally I would also include Windows 98 Second Edition in this list. There are a surprising number of people still running this OS; however, since Microsoft placed it into EOL (End of Life) status on July 11th, 2006, I see little reason to continue to support it, at least in this scenario. Just because they placed it into EOL doesn't mean you shouldn't support it; however, it can become tricky.

Simply follow the directions in the previous section to create and install the testing images. You do not, however, want to Sysprep these images.

Installing the VMware Tools Script to Retrieve the Guest IP Address

Next, because you know at this point the end goal is integration into your development process, you can go ahead and install a script into the VMware Tools package that will help you out a little later. If you remember from Chapter 11, I wrote a small C# console application that retrieved the guest machine's IP address and added it to a guest-info variable in VMware Tools. This allowed you to access the IP address from within your continuous integration environment outside the virtual machine. You can go ahead and take care of that now before you make your final changes to the test image.

1. Now, instead of firing up Visual Studio, you can just whack this one out from the command line. This will be a C# v2.0 console application like the one from Chapter 11. Create a new text file named `vmSetIP.cs` and enter the following code.

vmSetIP.cs

```
using System;
using System.Collections.Generic;
using System.Text;

namespace vmGuestGetIP
{
  class Program
  {
    static void Main(string[] args)
    {
      string computerName = System.Net.Dns.GetHostName();
      string computerIP = ↵
        System.Net.Dns.GetHostEntry(computerName).AddressList[0].ToString();
      setVMwareGuestInfo(computerIP);

    }

    static private void setVMwareGuestInfo(string computerIP)
    {
      try
      {
        System.Diagnostics.ProcessStartInfo proc = new
        System.Diagnostics.ProcessStartInfo("C:\\Program ↵
          Files\\VMware\\VMware Tools\\VMwareService.exe");
        proc.Arguments = "-cmd \"info-set guestinfo.ip " + computerIP + "\"";

        proc.WindowStyle = System.Diagnostics.ProcessWindowStyle.Hidden;
        proc.UseShellExecute = false;

        System.Diagnostics.Process GetIP;
        GetIP = System.Diagnostics.Process.Start(proc);
        GetIP.WaitForExit(3000);
      }
      catch (Exception e)
      {
```

(continued)

vmSetIP.cs *(continued)*

```
            Console.WriteLine("Unable to execute: " + e.Message);
          }
        }
      }
    }
```

2. Once you have that entered, go ahead and save and close and then open up a command-line window. To compile this C# application from the command line you need only the .NET Framework v2.0 installed. The C# compiler is located in the `C:\Windows\Microsoft.NET\Framework\<framework version>` directory. From the command prompt navigate to the directory in which you saved the `vmSetIP.cs` file and enter the following command:

```
C:\Windows\Microsoft.NET\Framework\<version>\csc.exe vmSetIP.cs
```

   As long as you entered the preceding code correctly, you should simply be returned a new console application named `vmSetIP.exe` in the current directory.

3. Copy this application to the `C:\Program Files\VMware\VMware Tools` directory on both of your testing images and open the VMware Tools application by double-clicking the icon shown in Figure 12-31.

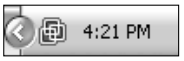

Figure 12-31

4. Once this opens you will find several tabs. The only one you are interested in is the Scripts tab. Click it to reveal Figure 12-32. From here you can set up VMware Tools to run your console application every time the guest powers on.

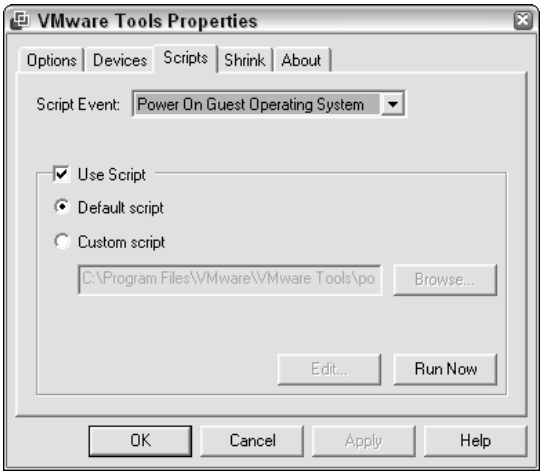

Figure 12-32

Click the Script Event drop-down list and choose Power On Guest Operating System. Make sure Use Script is checked and select Custom Script from the radio selections. Then click Browse to see Figure 12-33.

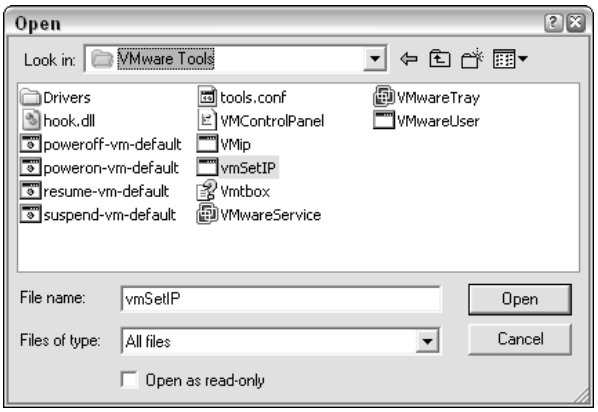

Figure 12-33

5. The `vmSetIP.exe` application you just copied over should be in the center pane. Double-click the console application, and the screen reverts back out to the one shown in Figure 12-34. Verify your settings are correct as shown in this figure and click OK to exit the dialog. This will conclude your test image setup for the moment. Go ahead and shut down the guest machine now.

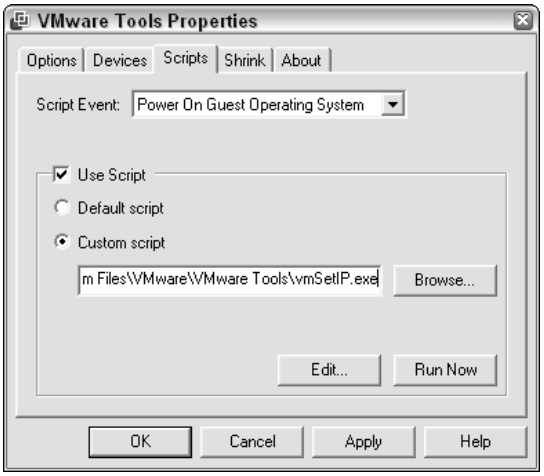

Figure 12-34

Configuring Your Testing Images as Nonpersistent

Now, if you think back to Chapter 5 where I discussed persistent and nonpersistent virtual disks, you already suspect this might be a good place to implement them. As a brief refresher, the normal default state of a virtual disk is *persistent*. This means that all changes performed during the normal operation of the virtual machine are permanently written to the virtual disk. A *nonpersistent* virtual disk is just the opposite. The virtual machine operates normally; however, once powered off all changes from that session are lost, and the next reboot provides a clean virtual machine in its original state. Almost all of my testing images in production are nonpersistent in nature. And you're going to make the two testing images in this example nonpersistent as well.

1. After you create and install the operating systems to the two testing images for this scenario, power them down and from the VMware Server Console select one of the two machine's tab from the main pane shown in Figure 12-35.

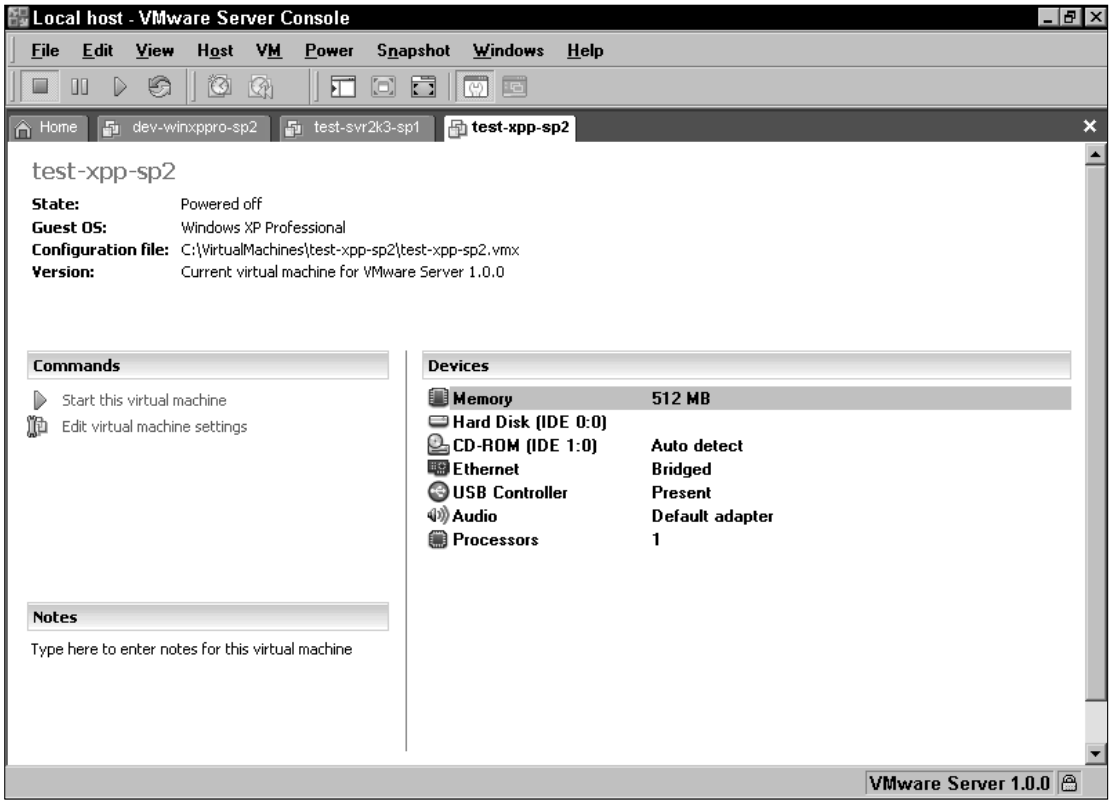

Figure 12-35

2. From here select the Edit virtual machine settings link from the main pane. Then, select the hard disk from the left-hand pane and click Advanced in the lower-right corner. This should result in Figure 12-36.

Figure 12-36

3. Place a check in Independent and select Nonpersistent from the radio selections. To ensure you're on track, you can compare with Figure 12-37.
4. Click OK and you'll find yourself back out at the virtual machine tab.

That's it for one of the test images. Repeat the process for the other test image in the example, and you'll be ready to move into the next stage.

Figure 12-37

Configuring Build Integration

Because Visual Studio Team System isn't technically a continuous integration product, I'm choosing my words carefully here and simply referring to it as build integration. No matter how you slice it, the end result is still the same — integrating VMware Server into your development process.

Of course before you can even get to this point you need a project. I'm not going to dive into creating a project in Visual Studio and pushing it into source control. I will, however, review the goals for this part of the scenario. The application is a C# WinForms application developed in Visual Studio 2005. How or what this application contains is of little importance for the end result. Just create any project and use Visual SourceSafe (VSS) for your source control system. That will satisfy the requirements just fine.

As before, the project will be named `ProVMwareSampleApp`. The location for this application in terms of VSS will be `\\vmserver-01\vss$\ProVMwareSampleApp`.

Project Setup

Now, it's time to turn to the build integration system. I believe the best choice for this scenario is CruiseControl.NET (CCNet). It's quick, easy to install, and simple to configure. You can start by installing CruiseControl.NET on your server.

Installing CruiseControl.NET on Your Server

Jump out to the CruiseControl.NET web site at `ccnet.thoughtworks.com` and grab the currently released installation files. At the time of writing the current release was 1.0.1. The site will redirect you to the downloads on `SourceForge.net`. There are several packages to choose from but to keep things simple just grab the `CruiseControl-1.0.1-Setup.exe` file and place it onto your server.

1. Once the file has downloaded go ahead and kick off the installer. After the standard welcome and agreement screens you should be looking at something similar to Figure 12-38.

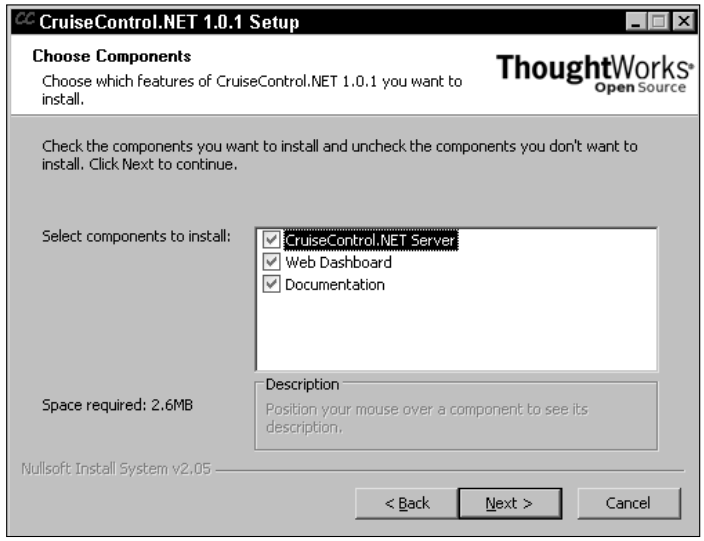
CruiseControl.NET 1.0.1 Setup
Choose Components
Choose which features of CruiseControl.NET 1.0.1 you want to install.
ThoughtWorks Open Source
Check the components you want to install and uncheck the components you don't want to install. Click Next to continue.
Select components to install:
CruiseControl.NET Server
Web Dashboard
Documentation
Description
Position your mouse over a component to see its description.
Space required: 2.6MB
Nullsoft Install System v2.05
< Back
Next >
Cancel

Figure 12-38

2. The choices here are obviously up to you; however, I suggest you leave them as is. At a minimum you need the CruiseControl.NET Server component. Click Next.

3. This brings you to a screen with a couple of configuration items, shown in Figure 12-39.

   Installing the CruiseControl.NET server as a service is a judgment call. As you are configuring the server, it is helpful to have it running in a console window so you can monitor the error output directly. For that reason the installer configures the service as manual so it will not start up on its own. Once it is part of the production environment, running it as a service is the better choice, and you can configure the service for automatic start through the control panel.

Also, go ahead and have the installer configure the IIS virtual directory. It just saves you the trouble later.

4. The final two screens of the installer are simply the directory and start menu group you wish to use. They are no different than any other installer in that regard.

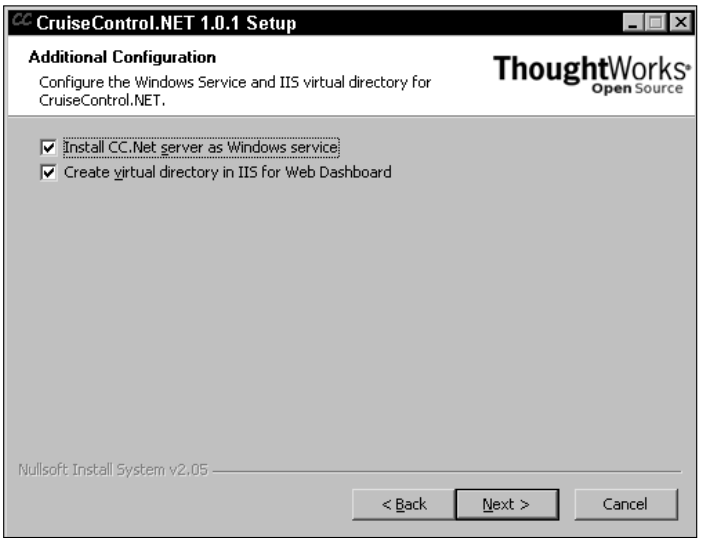
CruiseControl.NET 1.0.1 Setup
Additional Configuration
Configure the Windows Service and IIS virtual directory for CruiseControl.NET.
ThoughtWorks
Open Source
Install CC.Net server as Windows service
Create virtual directory in IIS for Web Dashboard
Nullsoft Install System v2.05
< Back
Next >
Cancel

Figure 12-39

Once the install completes, you will have a new icon on your desktop for the CruiseControl.NET server. This is how you will launch CruiseControl.NET during your testing.

Installing the CruiseControl.NET Tray Application on Your Development Machine

The next step to accomplish is to install the CCNet Tray application on our development machine:

1. Go ahead and start up the CCNet server application by double-clicking the icon the installer placed on your desktop. Once it is running, browse to the web dashboard at `http://localhost/ccnet`. You should receive the screen shown in Figure 12-40.

2. Click the Download CCTray link provided in the left column and install the application. The setup is pretty standard so just accept the default and continue through the install. Once complete you should be greeted with Figure 12-41, the CCTray main screen. You'll get into what you do with this in a bit. For now take note of it and close the window.

Now that you are done with that, you can move on to the CruiseControl.NET configuration file.

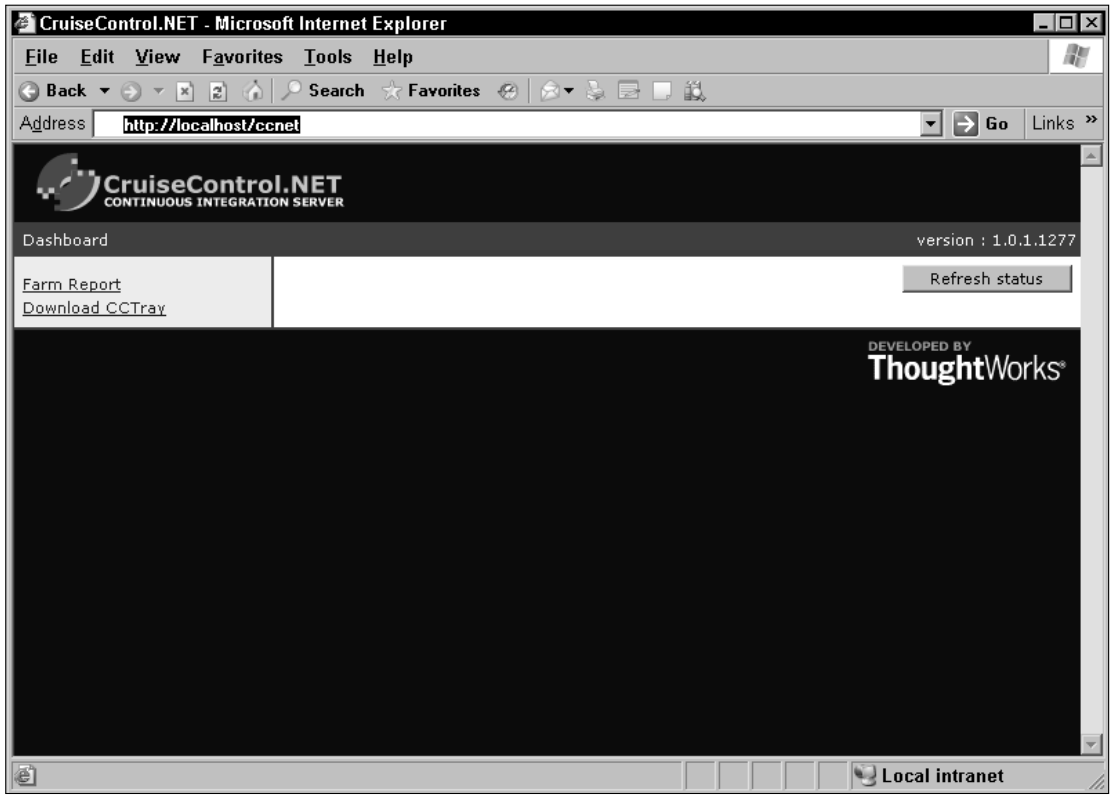

Figure 12-40

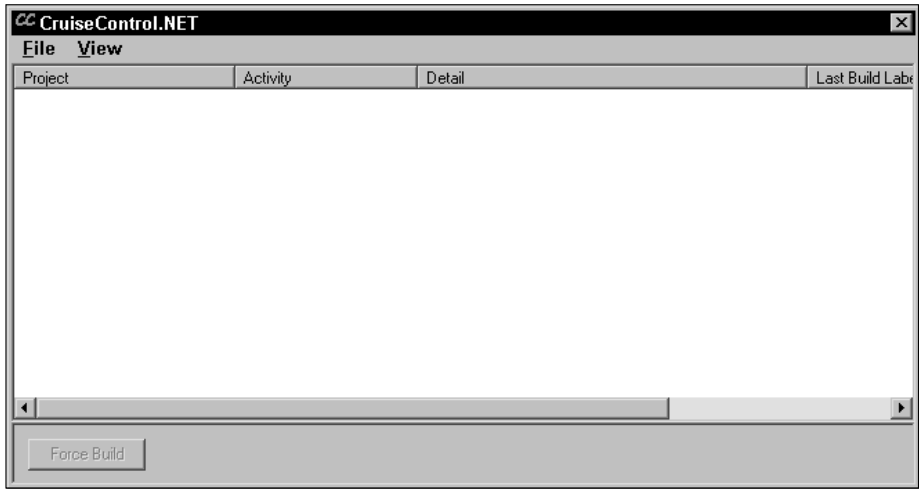

Figure 12-41

Configuring the CruiseControl.NET Configuration File

The key to setting up CCNet is in getting comfortable with the `ccnet.config` file. You can open it either through a convenient link in the Start menu or by browsing to its location in `C:\Program Files\CruiseControl.NET\server\ccnet.config`. There are several different items you can configure in the configuration file; however for this example you will be concentrating on this list:

- State
- Labelers
- Source controls
- Triggers
- Build tasks
- Executable tasks

First, you have to configure your root project node. Open your `ccnet.config` file, clear out the comments at the top, and enter the following between the root `<cruisecontrol>` nodes:

```
<project>
  <name>ProVMwareSampleApp</name>
  <workingDirectory>C:\CCNetWorking</workingDirectory>
  <artifactDirectory>C:\CCNetWorking\artifacts</artifactDirectory>
</project>
```

Save the configuration file now and switch over to your console window where CCNet is running. It should have picked up the change and state that there is nothing to do for project `ProVMwareSampleApp`. This is a good message because in fact you haven't told it to do anything yet. Also, just to come full circle, open up Internet Explorer and browse to the CCNet web dashboard. You should see a page very similar to Figure 12-42. If you experience an exception stating the tick counts are invalid, make sure you have the .NET Framework v2.x installed and configured for IIS.

Basically, the configuration data you've added so far includes the working directory node, which signifies where you would like CCNet to place files that are checked out for the build, and the artifact directory, which is where it will store its build logs.

> *These are optional; however, if you do not specify where to place the working and artifact directories, CCNet will place them as a subdirectory under* `C:\Program Files\CruiseControl.NET\Server`*, and personally I don't care for that setup. In this case I dislike the default and want my files kept in specific named directories for each type.*

Configuring State File Location

There's one more default that I dislike, and that's the location of the state file. Every project in CCNet has a corresponding state file. This file contains the specific configuration information for the project as well as the current build number, last successful build, and so on. By default this file lives in the `C:\Program Files\CruiseControl.NET\server` directory, and that's just not the best place for it in my opinion. Add the following block directly under the `<artifactDirectory>` node in your configuration file.

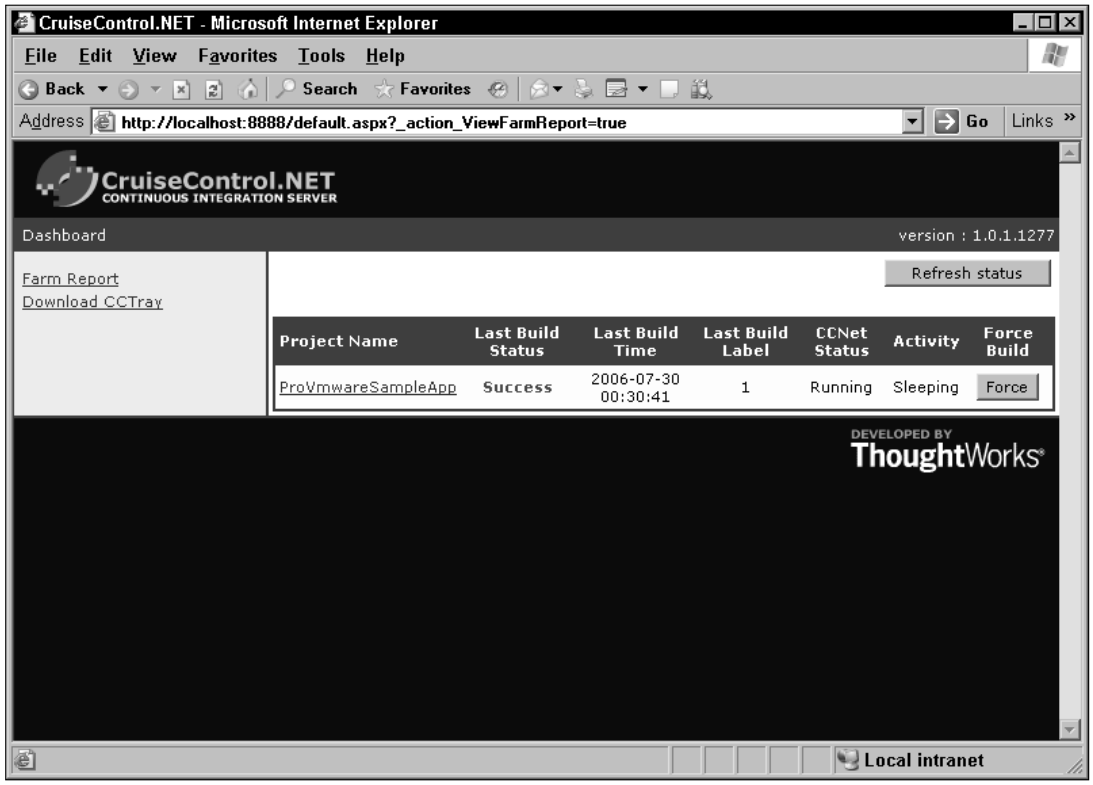

Figure 12-42

In contrast to the working and artifact directories, which CCNet creates automatically if they do not exist, the state directory is not created automatically by CCNet. You must create the state directory before you start up CCNet after making the change that follows. If you do not do this, CCNet will throw an exception that the directory does not exist.

```
<state type="state" directory="C:\CCNetState" />
```

Configuring the Labeler

Next is the labeler block. By default CCNet uses what it calls a *default labeller*. Left alone it will label each build by an incrementing number starting at one. Very dull and boring if you ask me. You can, however, configure a few different types of labelers for your project. This is not only the label that CCNet refers to the project as but also the label it applies to the source control checkout if you are using a source control provider that supports labeling.

I am going to use what's called an *iteration labeller*. I find this one a tad bit more interesting. The, dare I say, standard way of versioning would be *<major>.<minor>.<build>*. I think this is probably the scheme most use. The iteration labeler can do this for you. The idea here is that you provide it a major version number, or prefix, a duration of the iteration in weeks, and a start date for the project. It will take the prefix, append the iteration number, increment it every *x* number of weeks you set, and then follow it with the CCNet-generated build number.

Some may believe I'm spelling labeler wrong. That depends on who you ask; however I'm spelling it correctly in terms of CCNet. If your `labeller` *block is causing CCNet to throw an exception ensure that you spelled it with two l's.*

Enter the following code directly under the `<state>` block you added previously:

```
<labeller type="iterationlabeller">
  <prefix>1.0</prefix>
  <duration>1</duration>
  <releaseStartDate>2006/7/1</releaseStartDate>
</labeller>
```

Save your configuration file, and if CCNet is not running, start it up and browse to the web dashboard. Figure 12-43 shows you what mine looks like after the labeler node was added. You'll notice that the project start date was July 1, 2006, the prefix was set to 1.0, and the duration was 1 week. That results in a label of 1.0.4.1, meaning 4 weeks from the project start date (since today is July 30th), and CCNet build number 1. That's much better than just the boring old default and most likely comes closer to conforming to your current build number specifications.

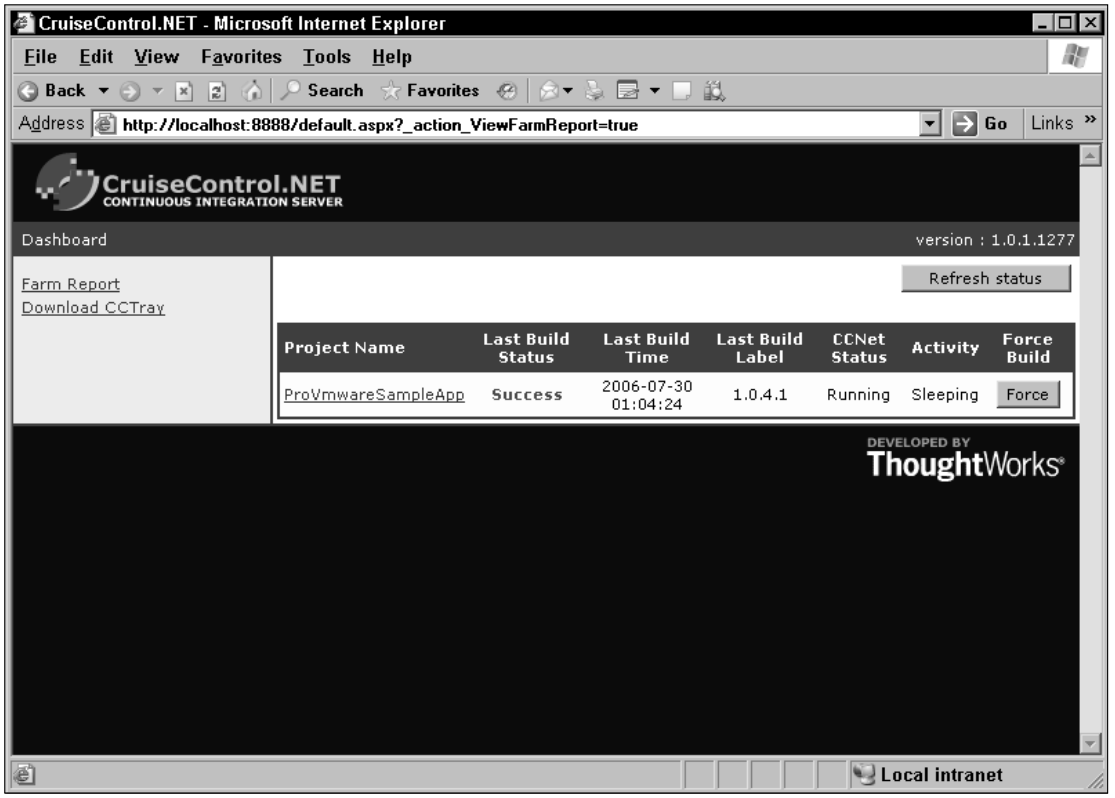

Figure 12-43

Configuring Source Control

Now you come to the source control node. Here you are given a plethora of options including some source control providers you may never have never even heard of. For a complete list of what's supported visit the CCNet web site at `ccnet.thoughtworks.com`. Because this scenario is utilizing Visual SourceSafe you'll stick with it. Enter the following directly after the `labeller` node, and I'll explain it in a bit:

```
<sourcecontrol type="vss" autoGetSource="true" applyLabel="true">
  <executable>C:\Program Files\Microsoft Visual SourceSafe\ss.exe</executable>
  <project>$/ProVmwareSampleApp.root</project>
  <ssdir>C:\VSS</ssdir>
  <username>administrator</username>
  <password>Pass@word!</password>
  <cleanCopy>true</cleanCopy>
</sourcecontrol>
```

Starting at the top of the code block, you see that the `autoGetSource` attribute tells CCNet to automatically retrieve a copy of the source from source control. If you leave this at `false`, CCNet will work with a copy of the source that exists locally. Second is the `applyLabel` attribute. That one's not too hard to figure out. It will apply the label you set up in the `labeller` node previously to the source upon checkout. This comes in handy if a build fails so you can check out from a specific label to retrieve a copy of the source as it existed when the build failed.

> **Before I even start to explain the preceding code block I need to point out that the Visual SourceSafe client needs to be installed on the CCNet server. It seems obvious, but nonetheless it should be said.**

The next two attributes, `executable` and `project`, are pretty self-explanatory so I'll just skip over them.

The `ssdir` is the location of your `srcsafe.ini file`. This file lives in the root of your Visual SourceSafe database directory. My repository just happens to be local so `C:\VSS` works here; however, if your repository lives on a remote machine, a UNC path will suffice. Just make sure the security context in which the CCNet server process runs can access the remote directory.

The `username` and `password` attributes are just that, the user name and password used to connect to the SourceSafe database. The account you use here must have write access to the database or else the labeler will fail.

That's the source control block in a nutshell. If you have a project in source control, substitute the project name, ssdir, user name, and password I just covered with your own information and fire up CCNet. It should retrieve a copy of the source and report that the build succeeded. Of course nothing was built, but it should have gotten the source. You can check the `C:\CCNetWorking` directory as well as the web dashboard for the current status.

Configuring the Trigger Block

Now you can move on to the trigger block. A *trigger* in CCNet is how you cause a project's build to get fired. For instance, say you have a project configured that depends upon another project's successful

build in CCNet. The dependent project would use what's called a *project trigger* to poll CCNet and determine the build status of the parent project. If it was successful, it would trigger a build of the dependent project. Make sense? It's really quite useful for complex projects and build trees. For your use here, however, it's overkill. Because you have a single project, there's little need for anything other than either the default trigger or perhaps a scheduled one.

The default trigger if nothing else is configured is an *interval trigger*. This causes an integration cycle to occur every 60 seconds and build only if a modification exists. Personally I find every 60 seconds a bit much; however, others may not. You can configure the default interval to any amount of seconds, and you can also force a build even if no changes were detected. In its default state, if configured, it would look like the node that follows:

```
<intervalTrigger name="IntervalTrigger" seconds="60" ↩
  buildCondition="IfModificationExists" />
```

Another trigger option is the `scheduleTrigger`. This allows you to set a time of day and day, or days, of the week to perform a build. This is handy for nightly builds or a special distribution build, like the one you are performing here. Because your goal here is to push a build out to a virtual machine, you certainly wouldn't want to do that every 60 seconds, so a `scheduleTrigger` is a good choice here. Enter the following into the configuration file directly after the `sourcecontrol` node. All triggers must be enclosed within a `<triggers>` block:

```
<triggers>
  <scheduleTrigger time="23:00" buildCondition="ForceBuild" />
</triggers>
```

CCNet will now perform an integration only once per day at 11:00 p.m., and it will force a build even if no modifications are detected. This will work well for this scenario.

Forcing a Build

There's a catch here after you set up the preceding trigger. Now CCNet will perform a build only once a day. This means that in order for you to continue testing your configuration you need to force a build. This is where the CCTray application comes into play.

Because you already installed the CCTray application you should have an icon like the one shown in Figure 12-44 in your taskbar.

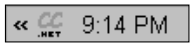

Figure 12-44

1. Make sure the CCNet server is running and double-click on the icon to reveal Figure 12-45. This is the main interface for the CCTry application.
2. Click File➪Settings to show the settings dialog in Figure 12-46. This is where you configure what server the CCTray application connects to and what projects it shows.

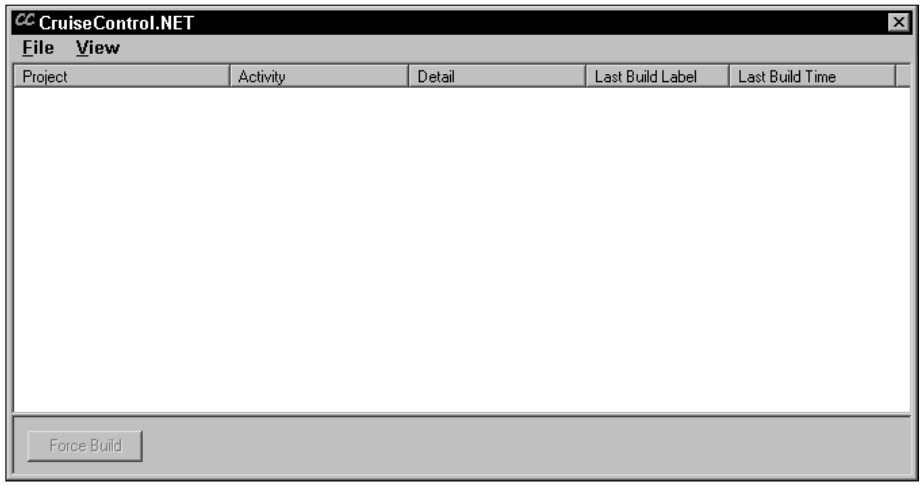

Figure 12-45

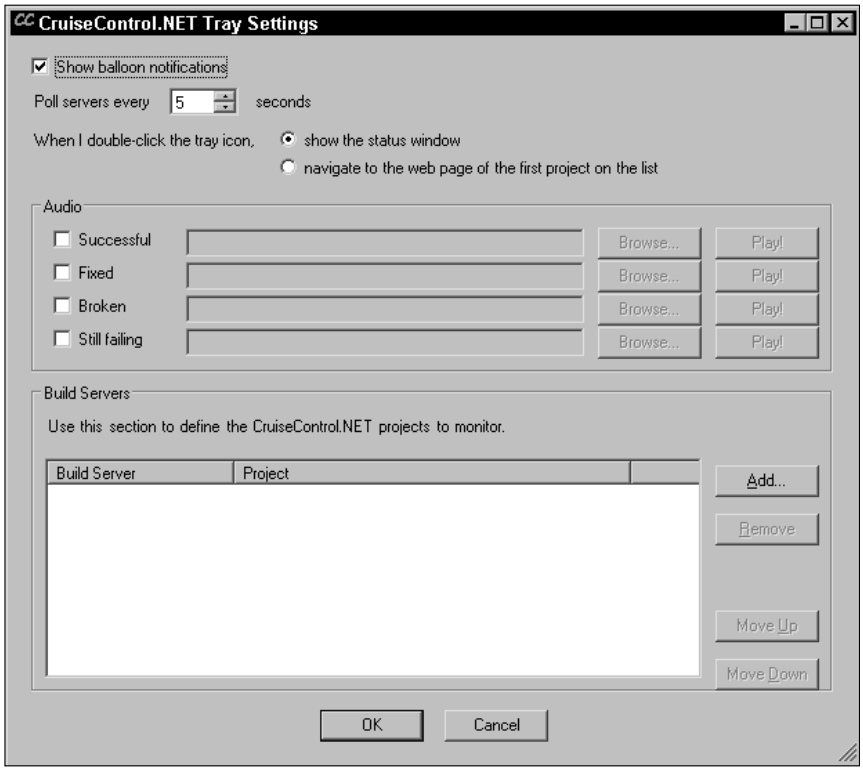

Figure 12-46

3. Click the Add button under the Build Server section. This shows the Project dialog. Under the Build Server section enter the name of your CCNet server; in this case it's `vmserver-01`. Click Add Server, and as long as your CCNet server application is running, it will populate the server and project panes of the dialog. Click to select the project on the right side, as shown in Figure 12-47.

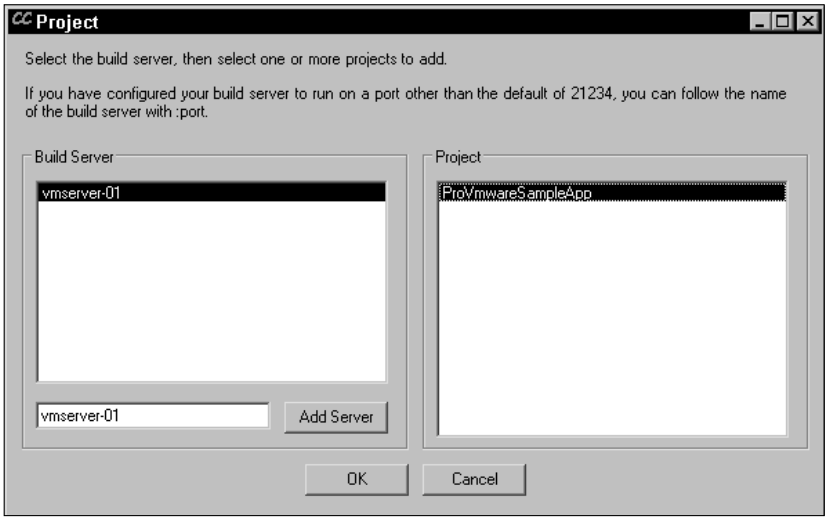
Project
Select the build server, then select one or more projects to add.
If you have configured your build server to run on a port other than the default of 21234, you can follow the name of the build server with :port.
Build Server
vmserver-01
vmserver-01
Add Server
Project
ProVmwareSampleApp
OK
Cancel

Figure 12-47

4. Click OK and verify the information was indeed passed onto the screen shown back in Figure 12-46. If all is well, click OK again to get back to the main CCTray interface. It should look similar to Figure 12-48.

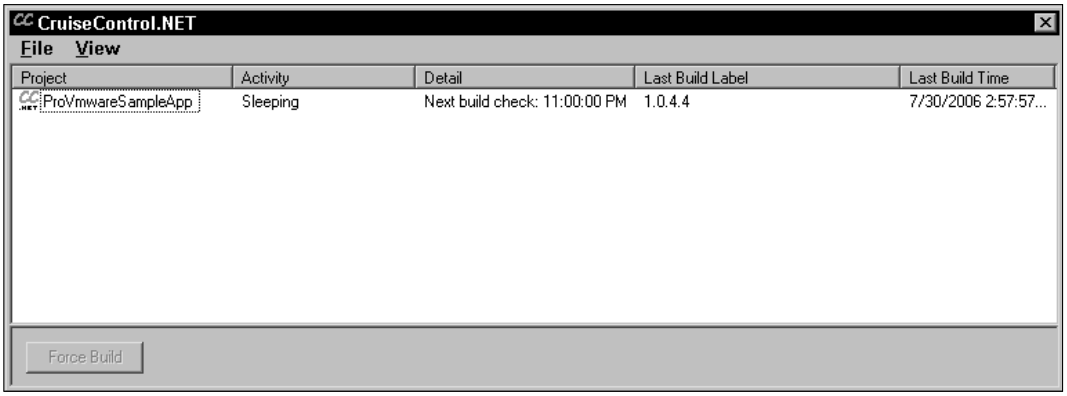

Figure 12-48

5. To force a build for a project, you need to have it in the CCTray application shown in the preceding figure. Right-click the project name to reveal a context menu like that shown in Figure 12-49. This is where you can force a project to build regardless of its integration schedule.

Figure 12-49

You can also force a build from the CCNet web dashboard shown in Figure 12-50. From the main screen click the Force button under the Force Build column for the project of interest. This will accomplish the same thing as the CCTray application.

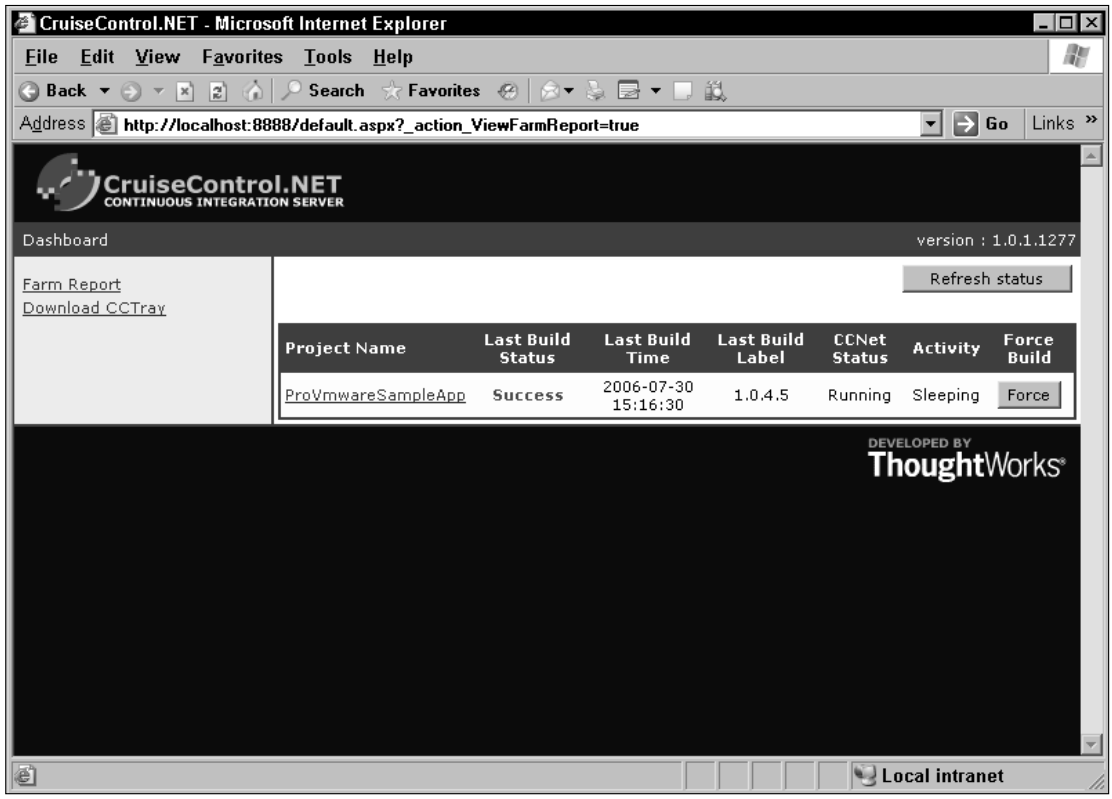

Figure 12-50

Configuring MSBuild

It's time to build something. Because your application is .NET based, has no unit tests, and is relatively simple I think MSBuild should do fine. MSBuild is installed with the .NET Framework v2.*x* automatically and therefore makes it a great choice. There is, however, a catch, just as there always seems to be. MSBuild doesn't come with an XML logger by default. To fully integrate MSBuild into CCNet, you have to perform two extra steps to provide an external XML logger:

1. First configure the MSBuild block. Enter the following directly after the `<triggers>` block. This, just like any build task in CCNet, must be wrapped in a `<tasks>` block.

```
<tasks>
  <msbuild>
    <executable>C:\WINDOWS\Microsoft.NET\Framework\v2.0.50727\MSBuild.exe
     </executable>
    <projectFile>C:\CCNetWorking\ProVMwareSampleApp\ProVMwareSampleApp.sln
      </projectFile>
    <buildArgs>/noconsolelogger /p:Configuration=Debug /v:diag</buildArgs>
    <timeout>15</timeout>
    <logger>ThoughtWorks.CruiseControl.MsBuild.XmlLogger, C:\Program
      Files\CruiseControl.NET\server\ThoughtWorks.CruiseControl.MsBuild.dll
      </logger>
  </msbuild>
</tasks>
```

   The executable node is rather self-explanatory; however, it should be noted that as .NET Framework versions change so will that path. Make sure you are referencing the proper version of MSBuild in your configuration file.

   Next the project file node refers to the solution file that exists in the project's working directory. You may have to actually navigate into the working directory to find this path, but it should point to the root project solution you are wanting to build.

   The build arguments node contains anything special you want to pass to MSBuild before it executes. You can refer to the MSBuild documentation for valid arguments, but the one I want to point out is the `/noconsolelogger`. This tells MSBuild not to output build information to the console. This keeps the CCNet console clear of the build output; however, it is still written to a file in the artifacts directory called `msbuild-results.xml`.

   Finally there's the logger node. This contains a reference to the custom XML logger you will be installing shortly. As I pointed out previously MSBuild doesn't come with an XML logger by default. This line tells CCNet to utilize an external logger for that purpose.

2. Now that your build configuration is set up, the next thing that you have to do is to grab the `ThoughtWorks.CruiseControl.MSBuild.XMLLogger` library from the CCNet web site at `ccnet.thoughtworks.com`. Certain restrictions disallow them from distributing it with the application; however, getting it is easy enough. If you jump on the CCNet web site and look under the documentation for the MSBuild task, you should find the link to download it easy enough. You need a file named `ThoughtWorks.CruiseControl.MsBuild.XmlLogger`. Once you download this file, save it into the CCNet server directory at `C:\Program Files\CruiseControl.NET\server`.

This is the same library you downloaded during the CCNet integration section of Chapter 11. You can refer to Chapter 11 for some specific search terms to help locate the file if necessary.

3. Next you have to configure the web dashboard to show a new menu selection for the MSBuild output. It's easy enough. Navigate to the `C:\Program Files\CruiseControl.NET\webdashboard` directory and open the `dashboard.config` file.

4. Scroll down until you find the `<buildPlugins>` node. From here, under the `<buildReportBuildPlugin>` section add the following `<xslFile>` under `<xslFileNames>`:

```
<xslFile>xsl\compile-msbuild.xsl</xslFile>
```

The XSL you just referenced is already present on your server. The `.dll` file shown earlier is the only missing piece, so don't worry.

5. Last, you'll need to add an `<xslReportBuildPlugin>` for MSBuild. This adds the menu option to select on the web dashboard. Add the following along with the other plugins:

```
<xslReportBuildPlugin description="MSBuild Output"
 actionName="MSBuildOutputBuildPlugin" xslFileName="xsl\msbuild.xsl" />
```

Save the `dashboard.config` file, and if CCNet isn't running, start it up and navigate to the web dashboard.

As long as you have a successful build with MSBuild, you should be able to view the most recent build's XML logger output. The dashboard will contain a new link on the left-hand side for MSBuild output, as shown in Figure 12-51.

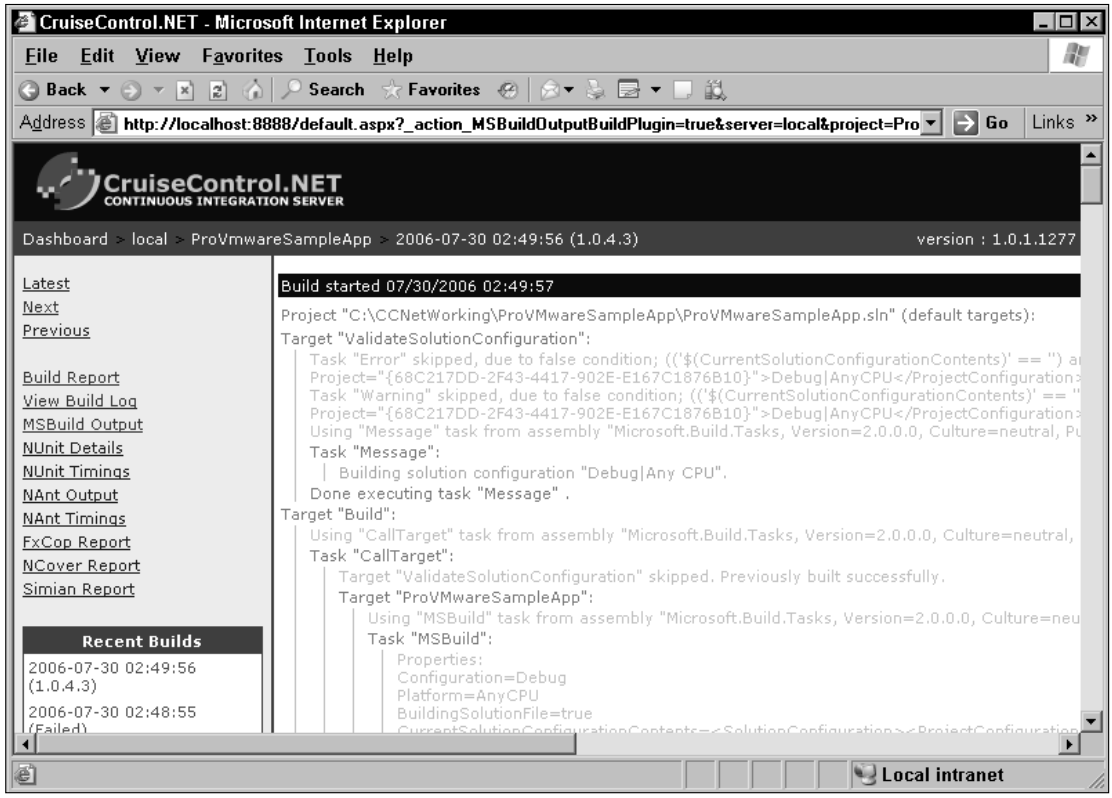

Figure 12-51

To view the screen shown in the preceding figure, you need to click the project name from the main dashboard page and then select the most recent build log on the left-hand side. At that point you can select the MSBuild output. You may need to force a build in order to get the results shown in the preceding figure.

Configuring the Build Publisher

Now you move on to the next member of the `<tasks>` block, the build publisher. Right now your project builds; however, the products of that build are buried deep within the project working directory. That's not a very handy place to have them so copy them out into a more common area.

Open the `ccnet.config` file again and add the following block within the `<tasks>` node directly after `</msbuild>`:

```
<buildpublisher>
  <sourceDir>C:\CCNetWorking\ProVMwareSampleApp\ProVMwareSampleApp\bin\Debug
   </sourceDir>
  <publishDir>C:\CCNetBuilds\ProVMwareSampleApp</publishDir>
</buildpublisher>
```

Just as was the case with everything except the `<state>` block, you will not have to create the `CCNetBuilds` directory yourself. Just open either the CruiseControl.NET Tray or web dashboard and force a build. Once it's complete, and successful, browse to the newly created `C:\CCNetBuilds` directory and verify the project directory and binaries are there. Piece of cake.

Configuring the External Script for VMware Server Integration

Last but not least comes the VMware Server integration. The details of that integration and the accompanying Perl script are in the next section; however, you need to set up the `ccnet.config` file with one final block under `<tasks>`. Go ahead and shut down the CCNet server because you do not want a build triggered during this phase of the configuration. Open the `ccnet.config` file and enter the following directly after the `<buildpublisher>` block but still within the parent `<tasks>` block:

```
<exec>
  <executable>perl</executable>
  <buildArgs>"C:\CCNetScripts\vmTestBank.pl xpp-sp2 svr2003-sp1"</buildArgs>
</exec>
```

This is the final step that is executed during the CCNet integration cycle. If you remember from Chapter 11, you call your Perl script at this point to integrate VMware Server into the equation. Because this is about a real-world scenario, however, you can spice it up somewhat. If you take a look at the `<buildArgs>` node, you'll notice that not only are you passing in the name of your Perl script but the types of testing machines you want to include. You use this information later in the chapter to fire up multiple virtual machines and push the builds out to each of them.

> *One other item to point out is the location of the Perl script. I've created yet another directory off the root called* `CCNetScripts`*. This is where I store common scripts that can be used across multiple CCNet projects.*

Save the `ccnet.config` file, and then you can move on to the VMware Server integration.

VMware Server Integration

You can move straight into some Perl. The basic characteristics of this script are as follows:

- Accept a list separated by spaces of the types of virtual machines to include
- Retrieve a list of virtual machines from the VMware Server

- ❑ Exclude the ones that do not match the criteria for the test
- ❑ Poll them for their status and if available add them into the available queue
- ❑ Power on the ones that are available and retrieve their IP addresses
- ❑ Connect to each machine and push the build up to them

That's quite a list, so you should take it step by step.

1. First, you create a new text file under the `C:\CCNetScripts` directory named `vmTestBank.pl`. Open it in your favorite text editor and get the following standard code out of the way:

```
#!/usr/bin/perl -w

# This is where you add the Windows specific libraries if the platform is MSWin32
BEGIN {
  if ($^O eq "MSWin32") {
    @INC = (
           'C:\Program Files\VMware\VMware VmPerl Scripting ↵
            API\perl5\site_perl\5.005',
           'C:\Program Files\VMware\VMware VmPerl Scripting ↵
            API\perl5\site_perl\5.005\MSWin32-x86');
  }
}

use strict;

# Import the VMware Perl modules needed
use VMware::VmPerl;
use VMware::VmPerl::Server;
use VMware::VmPerl::VM;
use VMware::VmPerl::ConnectParams;

# This extracts the integer value for constants out of the API
# Use this as a quick helper method to perform integer to constant string
# conversions
# This is adapted from one of the VMware samples and I use it in almost every
# script.
sub constant_to_string {
  my $constant_string = shift;
  return VMware::VmPerl::constant($constant_string, 0);
}

my $state_string_map = {};
my @state_strings = ("VM_EXECUTION_STATE_ON",
                     "VM_EXECUTION_STATE_OFF",
                     "VM_EXECUTION_STATE_SUSPENDED",
                     "VM_EXECUTION_STATE_STUCK",
                     "VM_EXECUTION_STATE_UNKNOWN");

# Loop through the strings above and pull out the integer value for each from the
# API
foreach my $state_string (@state_strings) {
  $state_string_map->{constant_to_string($state_string)} = $state_string;
}
```

This is pretty standard stuff for my VMware Perl scripts. I described it in detail in Chapter 11; however, for a quick review the basic idea is first to determine if it is running on Windows and if so to then load the appropriate libraries. Then, you fill the `$state_string_map` list with the values of the `VM_EXECUTION_STATE` constants to allow you to easily parse the state of each virtual machine.

2. Next, you want to connect to the server and grab a list of registered virtual machines. Enter the following directly after the preceding code:

```
my $server_name = "localhost";
my $port = 902;
my $username = "administrator";
my $password = "Pass\@word!";

# Create our $connectparams object to use when we connect to the server
my $connect_params = VMware::VmPerl::ConnectParams::new($server_name, $port,
                                                        $username, $password);

# Create the server object
my $server = VMware::VmPerl::Server::new();

# Connect to the server or display the error on failure
if (!$server->connect($connect_params)) {
  my ($error_number, $error_string) = $server->get_last_error();
  die "Count not connect to server: Error $error_number: $error_string\n";
}

# Grab a list of the available virtual machines in inventory and place them into an
# array
my @vm_list = $server->registered_vm_names();

# Make sure we actually have a list to use.
if (!defined($vm_list[0])) {
  my ($error_number, $error_string) = $server->get_last_error();
  die "Could not get a list of virtual machines from the server: Error
       $error_number : $error_string\n";
}

undef $server;
```

Aside from the error checking, the purpose of the preceding script is to simply fill the `@vm_list` list with the configuration file names of every registered virtual machine.

3. The next bit of code is where you grab your command-line arguments and parse the `@vm_list` list to acceptable virtual machines. If any are found, they are carted off into a new list named `@vm_matching`. Add the following to the end of your script:

```
if (!defined($ARGV[0]) {
  die "No machine specified on command line.\n";
}

my @vm_matching;

foreach my $vm (@vm_list) {

  foreach my $arg (@ARGV) {
```

```
    my $regex = "test-" . $arg . ".*";

    if ($vm =~ /$regex/) {
      push @vm_matching, $vm;
    }
  }

}

if (!defined($vm_matching[0])) {
  die "No matching virtual machines were found.\n";
}
```

Again passing over the error handling, if no command-line arguments were provided, you're into a bit of a double loop. You need to walk through the patterns provided on the command line, `@ARGV`, and then with each argument, walk through the list of registered virtual machines and find the matches. If the regular expression finds a match, you push the configuration file name, `$vm`, into the `@vm_matching` list.

Let me take a moment to explain the process of what's happening here. You pass in the pattern of a virtual machine type to test against. An example of one would be a Windows XP Professional with SP2. This is passed as simply `xpp-sp2`. Because the script is dealing with only test bank machines, there is little reason to pass the *test-* prefix into the script. You can just add it later. If your test bank machines utilized a project prefix along with the *test-* prefix, little modification is necessary in order to pass in that prefix and add it to the regular expression. Pop off the first argument, the prefix, and add it to the regular express `$regex`.

For example if you pass the following to the Perl script

```
perl vmTestBank.pl xpp-sp2 svr2k3-sp1
```

the loop would look through all registered machines using `test-xpp-sp2` as the regular expression string then loop through again searching for a match to `test-svr2k3-sp1`.

If you had a prefix to your test images, you could do the following:

```
Perl vmTestBank.pl 02 xpp-sp2 svr2k3-sp1
```

Then you would modify the previous code as follows:

```
my @vm_matching;
my $vm_prefix = shift @ARGV;

foreach my $vm (@vm_list) {

  foreach my $arg (@ARGV) {
    my $regex = "test-" . $vm_prefix . $arg . ".*";

    if ($vm =~ /$regex/) {
      push @vm_matching, $vm;
    }
  }

}
```

The modified version takes the first argument passed in and shifts it out of the `@ARGV` list into the scalar `$vm_prefix`. Then you add `$vm_prefix` into the regular expression. The regular expression would now attempt to find a match to *test02-xpp-sp2* and *test02-svr2k3-sp1*. The example here isn't that complicated, so I'll stick with the iteration without the prefix.

4. Next, you connect to each machine listed in `$vm_matching` and poll its current `VM_EXECUTION_STATE` to see if it's available. If it is, then you push that configuration file name into the `@vm_available` list. Enter the following directly after the preceding code:

```
my @vm_available;

foreach (@vm_matching) {
  my $vm = VMware::VmPerl::VM::new();

  if (!$vm->connect($connect_params, $_)) {
    my ($error_number, $error_string) = $vm->get_last_error();
    die "Could not connect to the virtual machine: Error $error_number :
         $error_string\n";
  }

  my $power_state = $vm->get_execution_state();

  if (!defined($power_state)) {
    my ($error_number, $error_string) = $vm->get_last_error();
    die "Could not get the virtual machine's power state: Error $error_number :
         $error_string\n";
  } elsif ($power_state == VM_EXECUTION_STATE_OFF) {
    push @vm_available, $_;
  }

  undef $vm;

}

if (!defined(@vm_available)) {
  die "No virtual machines are available that meet the criteria for this
       test.\n";
}
```

5. Now, here it gets a little tricky. Because you are utilizing multiple possible test machines, you need to fire each of them up individually and grab their IP address before you move on. In previous scripts where you only dealt with a single machine this could be accomplished in a single block. Now that you are dealing with multiple machines you have to turn each of them on and grab their IP address as you go. You are going to accomplish this with a hash.

   Enter the following at the end of your script:

```
my %vm_selected;

foreach (@vm_available) {
  $vm_selected{$_} = "IP";
}

foreach (keys %vm_selected) {
```

```
    my $vm = VMware::VmPerl::VM::new();

    # Connect to the VM
    if (!$vm->connect($connect_params, $_)) {
      my ($error_number, $error_string) = $vm->get_last_error();
      die "Could not connect to the virtual machine: Error $error_number :
           $error_string\n";
    }

    # Start up the VM
    if (!$vm->start(VM_POWEROP_MODE_SOFT)) {
      my ($error_number, $error_string) = $vm->get_last_error();
      die "Could not power on the virtual machine: Error $error_number :
           $error_string\n";
    }

    # Give the machine time to power on
    print "VM Starting, please wait\n";
    sleep(60);

    # Grab the IP from the VMware Tools Script
    my $ip = $vm->get_guest_info('ip');

    if (!defined($ip)) {
      my ($error_number, $error_string) = $vm->get_last_error();
      die "Could not get the IP address: Error $error_number :
           $error_string\n";
    } else {
      $vm_selected{$_} = $ip;
    }

  }
```

This one is a bit of a bear. What you do here is utilize a hash that contains the key value pair of configuration file name and IP address. At first you just fill the hash with the value of "IP." This allows you during development to see a change in the value when the IP is pushed in. It is, however, unnecessary in the grand scheme of things. The flow here is as follows: fill the hash with the items in `@vm_available` and then iterate through the hash keys turning on the virtual machines. Once the machine is powered on, grab the IP address from the VMware Tools script and add it to the value of that iteration's key. Pretty simple really. If you stripped out all the error handling it wouldn't be much code at all.

6. Finally, you move into the final phase of the Perl script, pushing the binaries up to the desktops. Again you are going to use an external call to "net use" here to map a drive so you can pass credentials, push up the files, and then kill off the mapped drive. You will be doing this for as many virtual machines as there are hash keys. Enter the following to the end of the Perl script:

```
foreach (values %vm_selected) {
 my @command = ("net use ",
                "x: ",
                "\"\\\\$_\\c\$\\Documents and Settings\\Administrator\\Desktop\" ",
                "/user:administrator ",
```

```
                "pass\@word!");

  system(@command);

  @command = ("xcopy ",
              "C:\\CCNetBuilds\\$ENV{CCNETPROJECT}\\$ENV{CCNETLABEL}\\*.* ",
              "x:\\$ENV{CCNETPROJECT}\\$ENV{CCNETLABEL}\\");

  system(@command);

  @command = ("net use ", "x: ", "/delete");

  system(@command);

}
```

This portion of the script assumes a few things that I want to point out. First and foremost the login and password to the machines must be the same in order for the preceding to work. Second is the Windows XP firewall and file sharing settings. By default Windows XP enables what it calls "simple file sharing." This must be turned off for the preceding to work. The reason is that simple file sharing doesn't create the administrative share `C$`. The firewall setting must also allow File and Printer sharing though.

- To disable simple file sharing on Windows XP double-click My Computer and select the Tools➪Folder Options menu, shown in Figure 12-52.

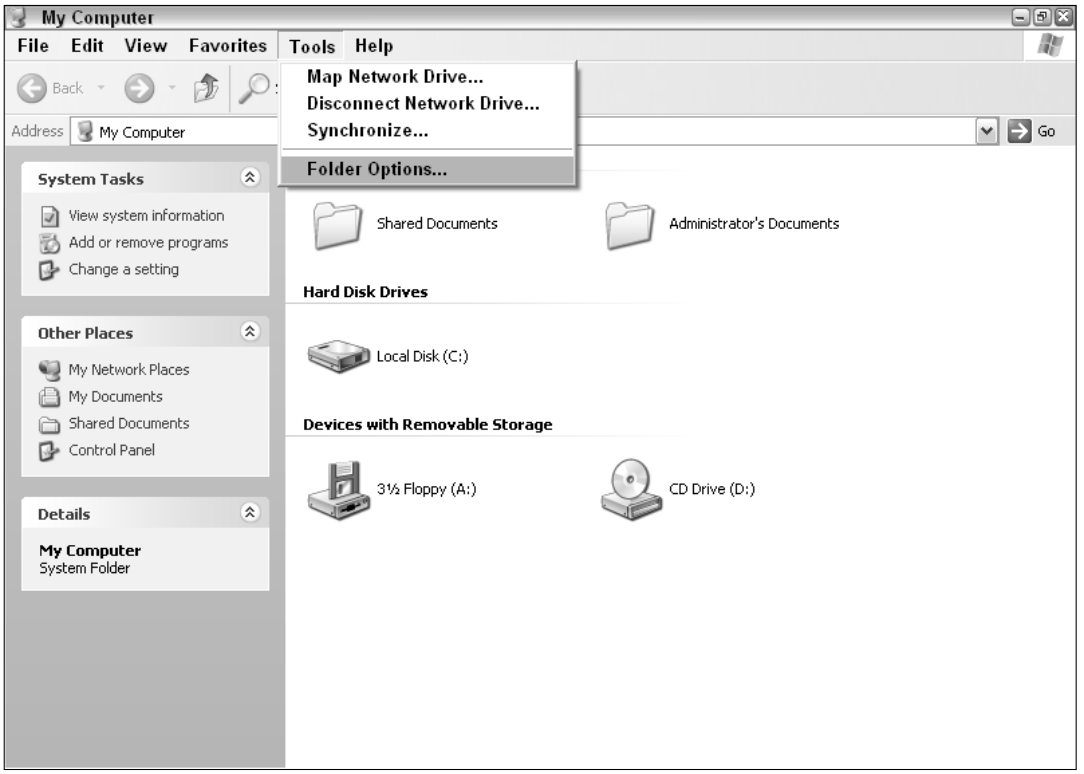

Figure 12-52

Once the form opens select the View tab and scroll all the way down the list under Advanced Settings. Here you need to remove the checkmark for Use simple file sharing, shown in Figure 12-53.

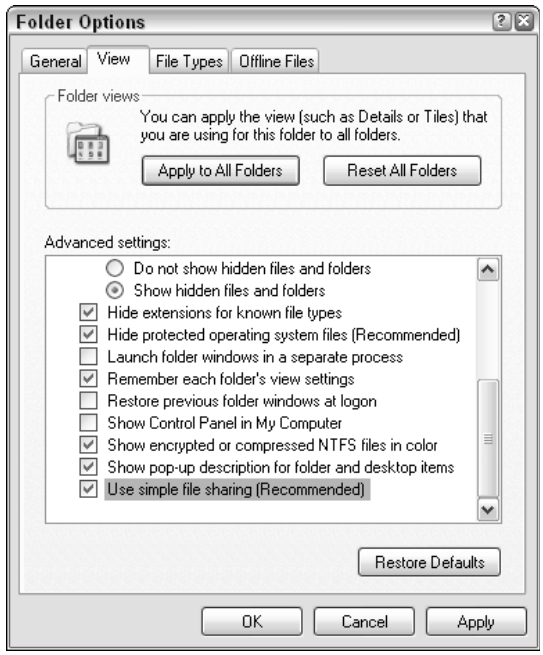
Folder Options
General
View
File Types
Offline Files
Folder views
You can apply the view (such as Details or Tiles) that you are using for this folder to all folders.
Apply to All Folders
Reset All Folders
Advanced settings:
Do not show hidden files and folders
Show hidden files and folders
Hide extensions for known file types
Hide protected operating system files (Recommended)
Launch folder windows in a separate process
Remember each folder's view settings
Restore previous folder windows at logon
Show Control Panel in My Computer
Show encrypted or compressed NTFS files in color
Show pop-up description for folder and desktop items
Use simple file sharing (Recommended)
Restore Defaults
OK
Cancel
Apply

Figure 12-53

Once this is done click OK to clear all of the dialogs and close My Computer.

- To allow File and Printer Sharing, right-click My Network Places or open your Network Settings from the control panel to show Figure 12-54. You want to right-click the network connection and choose Properties.

  Click the Advanced tab and select the Settings button under Windows Firewall. There will be three tabs. Select the Exceptions tab shown in Figure 12-55.

  Place a checkmark beside File and Printer Sharing and click OK to exit all the dialogs.

You will now be able to connect to the administrative share `C$` on the Windows XP machine.

Remember, however, that if you are doing this after you set your disk to nonpersistent, none of the changes will be saved. Make the disk persistent prior to starting it up to make this change permanent.

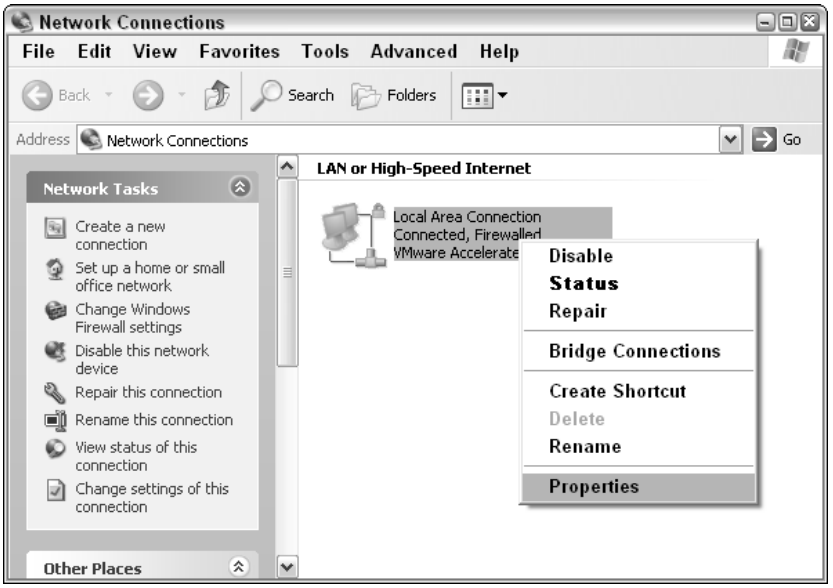

Figure 12-54

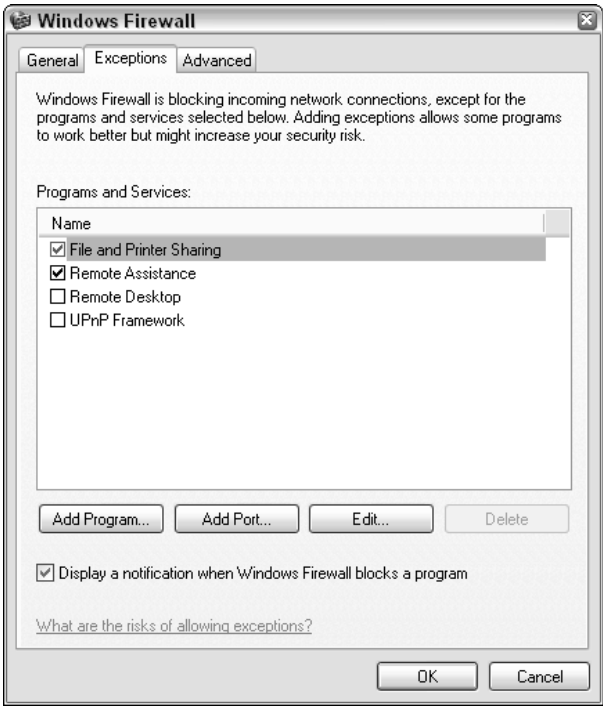

Figure 12-55

The Final Scripts

Now that you've completed the `ccnet.config` and `vmTestBank.pl` scripts, you can take a look at both in their entirety.

ccnet.config

```
<cruisecontrol>
  <project>
    <name>ProVmwareSampleApp</name>
    <workingDirectory>C:\CCNetWorking</workingDirectory>
    <artifactDirectory>C:\CCNetArtifacts</artifactDirectory>

    <state type="state" directory="C:\CCNetState" />

    <labeller type="iterationlabeller">
      <prefix>1.0</prefix>
      <duration>1</duration>
      <releaseStartDate>2006/7/1</releaseStartDate>
    </labeller>

    <sourcecontrol type="vss" autoGetSource="true" applyLabel="true">
      <executable>C:\Program Files\Microsoft Visual SourceSafe\ss.exe</executable>
      <project>$/ProVMwareSampleApp.root</project>
      <ssdir>C:\VSS</ssdir>
      <username>administrator</username>
      <password>Pass@word!</password>
      <cleanCopy>true</cleanCopy>
    </sourcecontrol>

    <triggers>
      <scheduleTrigger time="23:00" buildCondition="ForceBuild" />
    </triggers>

    <tasks>
      <msbuild>
        <executable>C:\WINDOWS\Microsoft.NET\Framework\v2.0.50727\MSBuild.exe➥
        </executable>

        <projectFile>C:\CCNetWorking\ProVMwareSampleApp\ProVMwareSampleApp.sln➥
        </projectFile>

        <buildArgs>/noconsolelogger /p:Configuration=Debug /v:diag</buildArgs>
        <timeout>15</timeout>
        <logger>ThoughtWorks.CruiseControl.MsBuild.XmlLogger, ➥
  C:\Program Files\CruiseControl.NET\server\ThoughtWorks.CruiseControl.MsBuild.dll
        </logger>
      </msbuild>

      <buildpublisher>
        <sourceDir>➥
          C:\CCNetWorking\ProVMwareSampleApp\ProVMwareSampleApp\bin\Debug➥
        </sourceDir>

        <publishDir>C:\CCNetBuilds\ProVMwareSampleApp</publishDir>
```

(continued)

ccnet.config *(continued)*

```
        </buildpublisher>

        <exec>
          <executable>perl</executable>
          <buildArgs>C:\CCNetScripts\vmTestBank.pl xpp-sp2 svr2k3-sp1</buildArgs>
        </exec>
      </tasks>

    </project>
  </cruisecontrol>
```

vmTestBank.pl

```
  #!/usr/bin/perl -w

  # This is where you add the Windows specific libraries if the platform is MSWin32
  BEGIN {
    if ($^O eq "MSWin32") {
      @INC = (
             'C:\Program Files\VMware\VMware VmPerl Scripting
              API\perl5\site_perl\5.005',
             'C:\Program Files\VMware\VMware VmPerl Scripting
              API\perl5\site_perl\5.005\MSWin32-x86');
    }
  }

  use strict;

  # Import the VMware Perl modules needed
  use VMware::VmPerl;
  use VMware::VmPerl::Server;
  use VMware::VmPerl::VM;
  use VMware::VmPerl::ConnectParams;

  # This extracts the integer value for constants out of the API
  # Use this as a quick helper method to perform integer to constant string
  # conversions
  # This is adapted from one of the VMware samples and I use it in almost every
  # script.
  sub constant_to_string {
    my $constant_string = shift;
    return VMware::VmPerl::constant($constant_string, 0);
  }

  my $state_string_map = {};
  my @state_strings = ("VM_EXECUTION_STATE_ON",
                       "VM_EXECUTION_STATE_OFF",
                       "VM_EXECUTION_STATE_SUSPENDED",
                       "VM_EXECUTION_STATE_STUCK",
                       "VM_EXECUTION_STATE_UNKNOWN");

  # Loop through the strings above and pull out the integer value for each from the
  # API
  foreach my $state_string (@state_strings) {
```

```
  $state_string_map->{constant_to_string($state_string)} = $state_string;
}

#####################################################################

my $server_name = "localhost";
my $port = 902;
my $username = "administrator";
my $password = "Pass\@word!";

# Create our $connectparams object to use when we connect to the server
my $connect_params = VMware::VmPerl::ConnectParams::new($server_name,
                                                        $port,
                                                        $username,
                                                        $password);

# Create the server object
my $server = VMware::VmPerl::Server::new();

# Connect to the server or display the error on failure
if (!$server->connect($connect_params)) {
  my ($error_number, $error_string) = $server->get_last_error();
  die "Count not connect to server: Error $error_number: $error_string\n";
}

# Grab a list of the available virtual machines in inventory and place them into an
# array
my @vm_list = $server->registered_vm_names();

# Make sure we actually have a list to use.
if (!defined($vm_list[0])) {
  my ($error_number, $error_string) = $server->get_last_error();
  die "Could not get a list of virtual machines from the server: Error
       $error_number : $error_string\n";
}

undef $server;

#####################################################################

if (!defined($ARGV[0])) {
  die "No machine specified on command line.\n";
}

# Walk through the list of vm's and grab the ones that match
my @vm_matching;

foreach my $vm (@vm_list) {
  foreach my $arg (@ARGV) {
    my $regex = "test-" . $arg . ".*";

    if ($vm =~ /$regex/) {
      push @vm_matching, $vm;
    }
  }
```

(continued)

vmTestBank.pl *(continued)*

```
}

if (!defined($vm_matching[0])) {
  die "No matching virtual machines were found.\n";
}

#########################################################################

# Connect to each one that matches and get its state
my @vm_available;

foreach (@vm_matching) {
  my $vm = VMware::VmPerl::VM::new();

  if (!$vm->connect($connect_params, $_)) {
    my ($error_number, $error_string) = $vm->get_last_error();
    die "Could not connect to the virtual machine: Error $error_number :
        $error_string\n";
  }

  my $power_state = $vm->get_execution_state();

  if (!defined($power_state)) {
    my ($error_number, $error_string) = $vm->get_last_error();
    die "Could not get the virtual machine's power state: Error $error_number
        : $error_string\n";
  } elsif ($power_state == VM_EXECUTION_STATE_OFF) {
    push @vm_available, $_;
  }

  undef $vm;
}

#########################################################################

my %vm_selected;

foreach (@vm_available) {
  $vm_selected{$_} = "IP";
}

foreach (keys %vm_selected) {

  my $vm = VMware::VmPerl::VM::new();

  # Connect to the VM
  if (!$vm->connect($connect_params, $_)) {
    my ($error_number, $error_string) = $vm->get_last_error();
    die "Could not connect to the virtual machine: Error $error_number :
         $error_string\n";
  }

  # Start up the VM
```

```
    if (!$vm->start(VM_POWEROP_MODE_SOFT)) {
      my ($error_number, $error_string) = $vm->get_last_error();
      die "Could not power on the virtual machine: Error $error_number :
           $error_string\n";
    }

    # Give the machine time to power on
    print "VM Starting, please wait.\n";
    sleep(80);

    # Grab the IP from the VMware Tools script
    my $ip = $vm->get_guest_info('ip');

    if (!defined($ip)) {
      my ($error_number, $error_string) = $vm->get_last_error();
      die "Could not get the IP address: Error $error_number : $error_string\n";
    } else {
      $vm_selected{$_} = $ip;
    }

  }

  ########################################################################

  foreach (values %vm_selected) {

   my @command = ("net use ",
                  "x: ",
                  "\"\\\\$_\\c\$\\Documents and Settings\\Administrator\\Desktop\" ",
                  "/user:administrator ",
                  "evyx5012!");

   system(@command);

   @command = ("xcopy ",
               "C:\\CCNetBuilds\\$ENV{CCNETPROJECT}\\$ENV{CCNETLABEL}\\*.* ",
               "x:\\$ENV{CCNETPROJECT}\\$ENV{CCNETLABEL}\\");

   system(@command);

   @command = ("net use ", "x: ", "/delete");

   system(@command);

  }
```

That's it! Put your scripts into place and fire off a forced build. If all goes well you should have a fully integrated CruiseControl.NET and VMware Server setup.

Summary

A real-world scenario is a difficult concept to put on paper. While no scenario is perfect nor can cover everything, hopefully what you have gained from this final chapter is a further understanding of the flexibility that the VMware Server APIs offer and their ability to conform to almost any situation. The example provided here, while similar in practice to previous chapters, expanded upon the basic ideas previously demonstrated and introduced some new techniques in which to accomplish your goals.

If you follow the pattern of defining your goals, setting up a very specific set of development and testing images, and planning out your level of integration, the implementation itself should prove to be rather simple.

A

VmCOM Error Codes

VMware has provided a simple yet effective way of detecting and handling errors in VmCOM, as is usually the case with most programming APIs. While somewhat limited in nature compared to the other API offerings, you are provided with two options for VmCOM:

- First every method and property in the VmCOM API offers up the standard error object, `Err`. The error object contains two properties for retrieving the error information and one method for clearing the error state. Success is indicated as a zero (0) for the error number. Take the following as an example:

```
if Err.Number <> 0 then
  WScript.Echo "An error has occurred: " & Err.Number & ": " & Err.Description
  Err.Clear
end if
```

  This is a pretty standard example of error handling within your VmCOM scripts. The preceding code would follow any server or virtual machine operation capable of throwing an exception. More often than not you would provide an `else` statement to perform some follow up if the operation were successful.

  In order for you to be successful in handling errors in this fashion, you need to provide the following directive at the beginning of every VmCOM script you write utilizing VB Script:

```
On Error Resume Next
```

  This tells the scripting runtime to not treat errors as fatal. Without this the script will simply exit when an error is thrown and not allow your error handling to take effect.

- The second way, which I will not be covering in detail, is by leveraging the `ISupportErrorInfo` interface. The standard object inherits from this interface itself. This means you could write a custom error-handling object inheriting from the `ISupportErrorInfo` interface to provide functionality beyond what the standard error object provides. For almost all situations this is a bit of overkill because the standard error

object can control the flow just fine on its own. Additionally, the documentation for this interface is currently very lacking. As the Vix API matures over the next several versions, the documentation for it will no doubt improve.

The following table lists the common VmCOM error codes and their descriptions. Note that the `Err` object exposes the following error codes as constant enumerations so it is completely valid to make the following statement:

```
if Err.Number = vmErr_NEEDINPUT then
  ...
end if
```

This error code list has been adapted from the official VMware Scripting Guide available from VMware. I've provided a version of it here to allow for quick reference while scripting.

Error Code	Description
`vmErr_NOTCONNECTED`	This error occurs if you attempt an operation that requires a connection to the server.
`vmErr_DISCONNECT`	This error occurs if the network connection to the host machine is lost or otherwise unreachable.
`vmErr_INVALIDARGS`	This error occurs if an argument provided to a method or property is invalid.
`vmErr_INVALIDVM`	This error occurs if the host machine cannot find the specified virtual machine.
`vmErr_NEEDINPUT`	This error occurs if the virtual machine referenced is currently in a STUCK state and cannot proceed. Most often you would follow with `VmCtl.PendingQuestion()`.
`vmErr_NOACCESS`	This error occurs when access to perform the requested operation is denied. This is usually due to the user you connect to the server with not having permission to manipulate the virtual machine referenced.
`vmErr_NOMEM`	This error occurs if the host machine has run out of available memory.
`vmErr_NOPROPERTY`	This error occurs if the referenced property name does not exist with the current object.
`vmErr_NETFAIL`	This error occurs if a network connection failure causes the referenced operation to fail.
`vmErr_INSUFFICIENT_RESOURCES`	This error can occur from any number of resource-related issues on the host machine. Most commonly it occurs if you reach the maximum allowed connection limit.

Error Code	Description
vmErr_PROXYFAIL	This error only occurs if you are using a proxy to connect to the VMware Server and it causes the operation to fail.
vmErr_NOTSUPPORTED	This error occurs if you attempt an operation not supported by the server you are currently connected to.
vmErr_TIMEOUT	This error occurs when a system standard timeout is reached while waiting for an asynchronous operation to complete.
vmErr_VMBUSY	This error occurs if you attempt to control a virtual machine that is currently in use on the server console.
vmErr_VMEXISTS	This error occurs when you attempt to register a virtual machine that is already present in the server's inventory.
vmErr_VMINITFAILED	This error indicates a problem during the initialization of the virtual machine on the host server.
vmErr_BADSTATE	This error occurs when you try to perform a state change on a virtual machine that is not supported. For example you attempt to power on a virtual machine that is already in the on state.
vmErr_BADVERSION	This error occurs if you attempt to use a version of the API that is incompatible with the server you are connecting to.
vmErr_UNSPECIFIED	This error is simply any unspecified error that occurs.

B

VmPerl Error Codes

Like VmCOM, the VmPerl API provides a convenient method to handle errors within your scripts. One difference, however, is that while the VmCOM API provides a single `Err` object that can be referenced regardless of the error type, the VmPerl API requires you to retrieve the error codes via methods specific to the object in question.

However, because the VmPerl API is made up of only two major objects this does not cause a problem. First, there's `$server object`. By utilizing the `$server->get_last_error()` method, you can retrieve a list containing the error number and description if the operation on the object failed. Take the following `connect` operation as an example:

```
my $server = VMware::VmPerl::Server::new();
my $connect_params = VMware::VmPerl::ConnectParams::new();

if (!$server->connect($connect_params)) {
  my ($error_number, $error_string) = $server->get_last_error();
  die "Could not connect to server: Error Number: $error_number :
$error_string\n";
}
```

This is a simple `connect` operation on a server. The flexibility of Perl allows the connection and error trapping to be wrapped up into a nice neat package. Because Perl interprets any non-zero return as true you can use `(!$server->connect($connect_params))` to evaluate the success or failure of the connection operation. You are negating the return with the `!` operator, so in pseudocode you are basically stating "if not connected."

Of course all of your operations are not against the `$server` object. Once you are connected to the server, you will more than likely start to manipulate various virtual machines utilizing the `$vm` object. Luckily the `$vm` object possess the same `get_last_error()`, which allows you to retrieve the error information like before. Take the following connection to a known virtual machine as an example:

```
my $vm = VMware::VmPerl::VM::new();

if (!$vm->connect($connect_params, $config)) {
  my ($error_number, $error_string) = $server->get_last_error();
  print STDERR "Could not connect to VM $config: Error $error_number: ".
               "$error_string\n";
}
```

The following table lists the common VmPerl error codes and their descriptions.

This error code list has been adapted from the official VMware Scripting Guide available from VMware. I've provided a version of it here to allow for quick reference while scripting.

Error Code	Description
VM_E_NOTCONNECTED	This error occurs if you attempt an operation that requires a connection to the server.
VM_E_DISCONNECT	This error occurs if the network connection to the host machine is lost or otherwise unreachable.
VM_E_INVALIDARGS	This error occurs if an argument provided to a method or property is invalid.
VM_E_INVALIDVM	This error occurs if the host machine cannot find the specified virtual machine.
VM_E_NEEDINPUT	This error occurs if the virtual machine referenced is currently VM_EXECUTION_STATE_STUCK and cannot proceed. Most often you would follow with `$vm->get_pending_question()`.
VM_E_NOACCESS	This error occurs when access to perform the requested operation is denied. This is usually due to the user you connect to the server with not having permission to manipulate the virtual machine referenced.
VM_E_NOMEM	This error occurs if the host machine has run out of available memory.
VM_E_NOPROPERTY	This error occurs if the referenced property name does not exist with the current object.
VM_E_NETFAIL	This error occurs if a network connection failure causes the referenced operation to fail.
VM_E_INSUFFICIENT_RESOURCES	This error can occur from any number of resource-related issues on the host machine. Most commonly it occurs if you reach the maximum allowed connection limit.
VM_E_PROXYFAIL	This error occurs only if you are using a proxy to connect to the VMware Server and it causes the operation to fail.
VM_E_NOTSUPPORTED	This error occurs if you attempt an operation not supported by the server you are currently connected to.
VM_E_TIMEOUT	This error occurs when a system standard timeout is reached while waiting for an asynchronous operation to complete.

Error Code	Description
VM_E_VMBUSY	This error occurs if you attempt to control a virtual machine that is currently in use on the server console.
VM_E_VMEXISTS	This error occurs when you attempt to register a virtual machine that is already present in the server's inventory.
VM_E_VMINITFAILED	This error indicates a problem during the initialization of the virtual machine on the host server.
VM_E_BADSTATE	This error occurs when you try to perform a state change on a virtual machine that is not supported. For example you attempt to power on a virtual machine that is already in the on state.
VM_E_BADVERSION	This error occurs if you attempt to use a version of the API that is incompatible with the server you are connecting to.
VM_E_UNSPECIFIED	This error is simply any unspecified error that occurs.

C

Vix Error Codes

The Programming API, also known as Vix, contains quite an extensive list of possible error codes. As such, you may find this list to be rather daunting; however, it is nonetheless valuable.

If you remember from Chapter 10, most of my examples used a rather simple approach to error trapping. Vix provides a specific object, `VixError`, that contains the necessary error information for you to leverage throughout your scripts. Only specific types of operations return an object of this type, so refer to Chapter 10 and the API documentation for information on the return type of the method you are using.

As an example the following attempts to connect to a host machine using the blocking method `VixJob_Wait()`. The evaluation of the `VixError` cannot take place until after the asynchronous method returns, and the error object is provided.

```
VixError err;
VixHandle hostHandle = VIX_INVALID_HANDLE;
VixHandle jobHandle = VIX_INVALID_HANDLE;
VixHandle vmHandle = VIX_INVALID_HANDLE;

jobHandle = VixHost_Connect(VIX_API_VERSION,
                            VIX_SERVICEPROVIDER_VMWARE_SERVER,
                            NULL, // hostName
                            0, // hostPort
                            NULL, // userName
                            NULL, // password,
                            0, // options
                            VIX_INVALID_HANDLE, // propertyListHandle
                            NULL, // callbackProc
                            NULL); // clientData

err = VixJob_Wait(jobHandle,
                  VIX_PROPERTY_JOB_RESULT_HANDLE,
                  &hostHandle,
```

```
                    VIX_PROPERTY_NONE);

  if (VIX_OK != err) {
    // Handle the error...
    goto abort;
  }
```

As you can see, the final `if` statement evaluates the `VixError` object for a successful status. An error code of `VIX_OK` tells you the operations completed successfully. If there was an error, you should ensure that you release the appropriate handles before exiting your script.

The following is a list of the current error codes possible with version 1 of the Vix API. Some are rather vague while others seem a bit repetitive. Remember this is a first-generation API so as it matures hopefully the error codes will clean up a bit.

> *The list provided has been adapted from the API documentation. Where appropriate and from experience I've included a description of the error. There are, however, some ambiguous error codes and vague descriptions. This is likely because of how new this API is.*

Standard Errors

Error Code	Description
`VIX_E_ANON_GUEST_OPERATIONS_PROHIBITED_ON_HOST`	Virtual machine operations are not allowed on this host by anonymous users.
`VIX_E_ANON_GUEST_OPERATIONS_PROHIBITED_ON_VM`	Virtual machine operations are not allowed on this virtual machine by anonymous users.
`VIX_E_BAD_VM_INDEX`	The index provided is invalid.
`VIX_E_CANCELLED`	The operation was canceled.
`VIX_E_CANNOT_CONNECT_TO_VM`	The connection to the referenced virtual machine failed.
`VIX_E_CANNOT_READ_VM_CONFIG`	The configuration file referenced cannot be read. It may not exist or be corrupt.
`VIX_E_DISK_FULL`	The host machine's disk is full.
`VIX_E_FAIL`	A generic internal error not covered by any other error.
`VIX_E_FILE_ACCESS_ERROR`	Permissions error with referenced file.
`VIX_E_FILE_ALREADY_EXISTS`	An attempt was made to use a file that already exists.
`VIX_E_FILE_ALREADY_LOCKED`	The referenced file is in use.
`VIX_E_FILE_ERROR`	A very generic file error occurred.
`VIX_E_FILE_NOT_FOUND`	The referenced file was not found.
`VIX_E_FILE_READ_ONLY`	Either the referenced file is marked read-only or access permissions prevent writing.

Error Code	Description
VIX_E_GUEST_OPERATIONS_PROHIBITED_ON_VM	Operations against the referenced virtual machine are not allowed. It is unclear what causes this error currently.
VIX_E_GUEST_OPERATIONS_PROHIBITED_ON_HOST	Operations against the referenced host are not allowed. Like the preceding error it is unclear what causes this error.
VIX_E_GUEST_USER_PERMISSIONS	An operation against a guest machine failed due to insufficient permissions.
VIX_E_HOST_USER_PERMISSIONS	An operation against a host server failed due to insufficient permissions.
VIX_E_INCORRECT_FILE_TYPE	The referenced file is not the type expected.
VIX_E_INVALID_ARG	One of the arguments passed is invalid for the current object.
VIX_E_INVALID_HANDLE	The reference provided is not of type `VixHandle`.
VIX_E_INVALID_PROPERTY_VALUE	The value provided to the referenced property is invalid.
VIX_E_INVALID_XML	The referenced file is invalid.
VIX_E_MISSING_ANON_GUEST_ACCOUNT	The account to be used for anonymous operations is missing from the configuration.
VIX_E_MISSING_REQUIRED_PROPERTY	A property required for the referenced handle is missing.
VIX_E_NO_GUEST_OS_INSTALLED	The referenced guest machine does not have a valid operating system installed.
VIX_E_NOT_FOUND	Unclear. The API documentation states that a section of a configuration file is missing.
VIX_E_NOT_SUPPORTED	The attempted operation is not supported for the current handle.
VIX_E_NOT_SUPPORTED_FOR_VM_VERSION	The attempted operation is not supported for the current virtual machine version. It is unclear if this is hardware or a VMTools version.
VIX_E_NOT_SUPPORTED_ON_HANDLE_TYPE	The attempted operation is not valid for the referenced handle type.
VIX_E_OBJECT_IS_BUSY	The handle is already in use by another operation.
VIX_E_OP_NOT_SUPPORTED_ON_GUEST	The attempted operation is not supported on the guest operating system.
VIX_E_OUT_OF_MEMORY	Out of memory. Assuming it's host server memory.
VIX_E_POWEROP_SCRIPTS_NOT_AVAILABLE	Possibly the VMTools power scripts were unable to execute.

Table continued on following page

Error Code	Description
VIX_E_PROGRAM_NOT_STARTED	An error occurred while attempting to start an application on the referenced virtual machine.
VIX_E_READ_ONLY_PROPERTY	Occurs when you try to set a property that is read-only.
VIX_E_ROOT_GUEST_OPERATIONS_PROHIBITED_ON_HOST	What makes this different from the other operation-prohibited errors is unclear. This refers to an administrative user according to the API documentation.
VIX_E_ROOT_GUEST_OPERATIONS_PROHIBITED_ON_VM	What makes this different from the other operation-prohibited errors is unclear. This refers to an administrative user according to the API documentation.
VIX_E_TEMPLATE_VM	An unsupported operation was attempted against a virtual machine marked as a template.
VIX_E_TIMEOUT_WAITING_FOR_TOOLS	The timeout expired waiting for VMware Tools to start on the guest machine.
VIX_E_TOO_MANY_HANDLES	There are too many handles currently open.
VIX_E_TOOLS_NOT_RUNNING	The VMware Tools package does not appear to be running on the guest machine.
VIX_E_TYPE_MISMATCH	The object provided is not of the expected type.
VIX_E_UNRECOGNIZED_COMMAND	The command provided is unrecognized.
VIX_E_UNRECOGNIZED_PROPERTY	The property provided is unrecognized.
VIX_E_VM_ALREADY_LOADED	It is unclear if loaded refers to registered or open.
VIX_E_VM_INSUFFICIENT_HOST_MEMORY	The host server does not have sufficient memory to power on the virtual machine.
VIX_E_VM_IS_RUNNING	The virtual machine is already running.
VIX_E_VM_NOT_ENOUGH_CPUS	This error occurs if you configure your virtual machine to use Virtual SMP, or multiple processors, and the host machine does not have more than one processor available.
VIX_E_VM_NOT_FOUND	The referenced virtual machine cannot be found. It is unclear if this is an inventory error or a registration error.
VIX_E_VM_NOT_RUNNING	This error occurs if you attempt an operation on a virtual machine that requires it to already be running.
VIX_OK	The operation was successful.

Snapshot Errors

Error Code	Description
VIX_SNAPSHOT_CONFIG	The snapshot configuration file appears to be corrupt or has some other unspecified error.

Error Code	Description
VIX_SNAPSHOT_DISKLIB	This appears to be some sort of disk error with respect to the snapshot files.
VIX_SNAPSHOT_DISKLOCKED	The disk or files are locked and cannot be used. It is unclear exactly what this error means.
VIX_SNAPSHOT_DUMPER	According to the API documentation this error means unable to open snapshot file.
VIX_SNAPSHOT_DUPLICATEDDISK	According to the API documentation this error means the virtual disk is used multiple times.
VIX_SNAPSHOT_EXISTS	The name provided for the snapshot file already exists.
VIX_SNAPSHOT_INCONSISTENT	The snapshot files are in an inconsistent state.
VIX_SNAPSHOT_INDEPENDENTDISK	A snapshot taken while the virtual machine is powered on and the virtual disk is configured as independent is not allowed.
VIX_SNAPSHOT_INVAL	Assuming a generic internal error.
VIX_SNAPSHOT_LOCKED	The snapshot file is locked due to a pending operation or other unspecified reason.
VIX_SNAPSHOT_NAMETOOLONG	The snapshot file name provided is too long.
VIX_SNAPSHOT_NOCHANGE	The snapshot has not changed.
VIX_SNAPSHOT_NONUNIQUE_NAME	The snapshot name provided does not uniquely identify it.
VIX_SNAPSHOT_NOPERM	Appears to be a permission denied error.
VIX_SNAPSHOT_NOTFOUND	The referenced snapshot file name does not exist.
VIX_SNAPSHOT_VERSION	According to the API documentation this means that snapshots are not allowed on this virtual machine.
VIX_SNAPSHOT_VIXFILE	According to the API documentation this means that it cannot snapshot all metadata files.

Teaming Errors

Error Code	Description
VIX_VM_IS_IN_VMTEAM	The virtual machine belongs to a team.
VIX_VM_IS_NOT_IN_VMTEAM	The virtual machine does not belong to a team.
VIX_VMTEAM_IS_IN_POWER_OP	The team is in the middle of a power operation.
VIX_VMTEAM_IS_RUNNING	The team is powered on.

Index

SYMBOLS & NUMERICS

~directory, as shortcut, 24
`/etc/services` **file, 51**
~modifier, 35
`$^O` (`$OSNAME`) **scalar, 184**
`$server®aget_last_error()` **method, 413**
#! (shebang), 153, 184
`/tmp` **folder, 115**
`$vm@@raget_last_error()` **method, 413–414**
64-bit platform, 353
902 port, verifying availability of, 50

A

accessing host settings, 62
adapter architecture, selecting, for virtual machine, 105–106
Add Reference screen (Visual Studio Team System), 331
Add Solution to Source Control screen (Visual Studio Team System), 320, 321
Add Team Foundation Server dialog box (Visual Studio Team System), 313
Add to Source Control dialog box (Visual Studio Team System), 346
Add to SourceSafe dialog box (Visual SourceSafe), 286–287
adding
  build distribution node, 298–299
  development project
    to source control, 285–288
    to team project, 318–322
  directory to Additional Include Directories window, 218
  source control, 289–291
  trigger, 288–289
Additional Dependencies option, 219, 220
Additional Include Directories option, 216
Additional Include Directories window, 217, 218
Add/Remove Team Foundation Server screen (Visual Studio Team System), 313, 314
Administering permission, 93
Advanced button (settings pane), 143
Advanced dialog box, 379, 380
Advanced section (Options tab), 82–84
Advanced tab (System Properties screen), 210, 211
Allocate all disk space now option (New Virtual Machine Wizard), 363
allowing File and Printer sharing, 401
AMD
  PAE standard and, 13
  processor requirements, 12
answer file
  creating, 369–372
  storing, 121
answers, default, to install queries, 26, 36, 45
antivirus software, 134
API (application program interface). ***See also*** **Programming API; scripting APIs; VmCOM API; VmPerl API**
  automation and, 151
  installing, 42
  supported platforms, 153–154
  tasks for automation, 152
Application Settings screen (Win32 Application Wizard), 214
Applications® System Tools, 33, 52
architecture, hosted versus hypervisor, 3–5
archiving
  base images, 140
  development environment, 127, 131
  Sysprep folder, 372
Ask me (Snapshots section), 77
asynchronous operations, 226–230
authenticating and logging in via code, 262–263
Autofit section (Display tab), 72
Automatic Bridging tab (Virtual Network Editor), 86
automation, tasks for, 152. ***See also*** **API**

B

bandwidth, 14
base image. *See also* development image; testing image
archiving, 140
creating, 98–109
description of, 97, 128, 138
freezing Windows guest, 116–123
installing operating system onto, 109–113
ISO image library, creating, 144–148
library of, organizing, 137–140
licensing issues, 98, 123
naming, 137–138
preparing for use, 140–141
batch file
placing in team project, 342–347
in VMware Tools, 281
binaries
copying to virtual machine, 337–338
obtaining prior to installation, 19, 24
pushing up to desktop, 399–400
BIOS, virtual, 74
blocking until job is complete, 228–230
bridged networking, 28–29
Browsing permission, 93
build
configuring, 296–299
forcing, 388–391
build distribution node, adding, 298–299
Build Explorer (Visual Studio Team System), 322
build integration, configuring
final scripts, 403–407
overview of, 380
project setup
CCNet Tray application, installing, 382–383
`ccnet.config` file, 383–394
CruiseControl.NET, installing, 381–382
VMware Server integration, 394–402
build log window (Visual Studio Team System), 329
build publisher, configuring, 394
Build Servers section (CruiseControl.NET Tray Settings screen), 292
build task, coding, 333–337
Build Team Build Project screen (Visual Studio Team System), 329
BusLogic adapter, 105

C

C compiler and building modules, 48
C#, scripting VMware Tools with, 283–284
`.cab` **file, 117, 368**
callback function
asynchronous operations, job handle, and, 230
`VixHost_FindItems()` function and, 240–243
CCNet (CruiseControl.NET). *See also* CCNet Tray application; `ccnet.config` file
Additional Configuration screen, 382
build, configuring, 296–299
build distribution log, adding, 298–299
configuring, 285–299, 383–394
continuous integration and, 278
Dashboard, 295–296, 391
description of, 151, 285
developmental project, adding to source control, 285–288
error handling, 302
installing, 381–382
IP address of virtual machine, retrieving, 305
main screen, 291, 294
parsing list of virtual machines, 302–304
powering on first available machine, 304
retrieving list of registered virtual machines, 300–302
source control, adding, 289–291
trigger, adding, 288–289
version of, 298
VMware Server integration, 299–311
XML logging, 298
CCNet Dashboard, 295–296
CCNet icon, 291
CCNet Tray application
configuring, 291–295
forcing build, 388–391
installing, 382–383
main interface, 388, 389, 390
Project dialog box, 390
Settings screen, 388, 389
CCNet Tray icon, 388
`ccnet.config` **file**
build publisher, 394
calling Perl script after build is completed, 310–311
categories of, 288
code, 310–311, 403–404
external script for VMware Server integration, 394
labeler, 385–386
overview of, 384
root project node, 384, 385
source control, 387
state file location, 384–385
trigger block, 387–388
`CCNetLabel` **variable, 306**
CD-ROM autorun, disabling, 20, 22
CD-ROM icon, 148
chip set, virtual, 74
code
authenticating and logging in via, 262–263
`ccnet.config` file, 310–311, 403–404
example1.pl, 188–190
example1.vbs, 163–164
example1.wsf, 163
example2.pl, 196–199
example2.vbs, 169–170
example2.wsf, 169
example3.pl, 200–203
example3.vbs, 172–173
example3.wsf, 172
example4.vbs, 175–176
example4.wsf, 174
`getIP.vbs` file, 280
`getIP.wsf` file, 280
`Program.cs` file, 283
`Program.vb` file, 284
`test-winxppro-sp2` file, 302–303

VMIntegration class, 338–342
vmSetIP.cs file, 375–376
vmTestBank.pl file, 307–310, 404–407
writing from within development virtual machine, 9
coding build task, 333–337
collaborating on development image, 133
COM (Component Object Model), 153, 155
commands
Create Launcher, 52–53
mount, 114
shutdown, 115
su -, 27
Commands area (main pane)
components of, 60
Edit virtual machine settings, 72
compiling against Vix with Visual C++
include statement and, 215
overview of, 209–210
pointing project
to header files, 216–218
to libraries used, 218–220
setting up project, 210–215
compiling kernel modules, 48–50
Completing Setup Manager screen (Setup Manager Wizard), 371–372
Component Object Model (COM), 153, 155
compressing base image for archiving, 140
compression utility, 25, 140
configuration file for virtual development machine, renaming, 365–367
Configure Host option (Home tab), 62
Configure Product screen, 22
configuring. *See also* ccnet.config file
build, 296–299
CCNet Tray application, 291–295
CruiseControl.NET (CCNet), 285–299, 383–394
Linux installation, 27–34
MSBuild, 296–299, 391–393
team build, 322–328
testing images as nonpersistent, 378–380
user access levels, 93–96
Configuring permission, 93
Confirm Team Project Settings screen (New Team Project wizard), 317
Connect to Team Foundation Server screen (Visual Studio Team System), 314
connecting to server
function example, 271–275
Perl script, 396
VixHost_Connect function and, 237–239
VmCtl object and, 169–171
Connection tab (host settings), 66, 67
connection to server, verifying, 187, 192
console preferences (server console)
Display tab, 71–72
Hot Keys tab, 70–71
Input tab, 68–69
overview of, 62, 66–67
Workspace tab, 68
constants, VmPerl API, 204–207
context menu (Linux), 52–53
continuous integration, 278, 349
Continuum continuous integration package, 349
copy and paste section (Input tab), 69
copying
base image, 134
binaries to virtual machine, 337–338
file
from guest machine to host machine, 266–267
from host machine to guest machine, 265–266
to virtual machine, 271–275
C:\\Program Files\VMware\VMware VIX **directory, 210**
Create Launcher command (Linux), 52–53
creating
answer file, 369–372
base image, 133–134
custom assembly for integration, 330–333
ISO image library, 144–148
snapshot, 251–252
team project, 312–318
temporary object, 236
virtual disk, 360
virtual machine, 98–109
vmware-users group, 95–96
cross-platform Perl scripts, 183–185
CruiseControl.NET (CCNet). *See also* CCNet Tray application; ccnet.config file
Additional Configuration screen, 382
build, configuring, 296–299
build distribution log, adding, 298–299
configuring, 285–299, 383–394
continuous integration and, 278
Dashboard, 295–296, 391
description of, 151, 285
developmental project, adding to source control, 285–288
error handling, 302
installing, 381–382
IP address of virtual machine, retrieving, 305
main screen, 291, 294
parsing list of virtual machines, 302–304
powering on first available machine, 304
retrieving list of registered virtual machines, 300–302
source control, adding, 289–291
trigger, adding, 288–289
version of, 298
VMware Server integration, 299–311
XML logging, 298
CruiseControl.NET Tray Settings screen, 292, 294
Ctrl+Alt+Del, 70
Ctrl+Alt+Enter, 71
Ctrl+Alt+F1 to F6, 114
Ctrl+Alt+Ins, 70
Ctrl+Alt+Space, 114
Ctrl+Shift+Alt+Ins, 70
cursor behavior, 69
custom assembly, creating for integration, 330–333
Custom Setup screen, 20, 21
Customer Information screen, 22

D

debugging mode, 84
default answers to install queries, 26, 36, 45
default hardware items and settings for operating system, 98–99
default hot key, changing, 70
default labeller, 385
default port, 31
defining
goals, 353–354
target environment, 351–353
deleting
virtual development machine floppy drive, 365
virtual machine, 250–251
`deploy.cab` **file, 116, 117**
deploying development image, 135
deployment, setting up virtual development machine software for, 367–374
depreciation of scripting APIs, 179, 180
desktop of virtual machine, pushing project payload to, 305–306
determining
number of root snapshots, 253–254
operational status, 204
results of pending job handle, 227–230
developer machine, 8
development environment
archive of, 127, 131
creating base images for, 131–135
integrating power of virtual machines into, 277
overview of, 125
recovery of, 126–127
virtual machines and, 126–127
development image
collaborating on, 133
creating, 131–135
deploying, 135
description of, 138
planning, 132–133
setting up, 354–364
testing, 134–135
development machine, virtual
deleting floppy drive, 365
renaming configuration file, 365–367
setting up, 355–364
setting up software for deployment, 367–374
development process, managing, 353
development server, virtualization of, 9
developmental project, adding
to source control, 285–288
to team project, 318–322
Device status screen, 110
Devices area (main pane), 60
Devices tab (host settings), 66
DHCP Settings window, 88–89
DHCP tab (Virtual Network Editor), 92
directory
adding to Additional Include Directories window, 218
shortcut (~), 24
working, 298
Disable acceleration option (Advanced section), 84
Disable memory page trimming option (Advanced section), 84
disabling. *See also* enabling
CD-ROM autorun, 20, 22
memory page trimming, 84
simple file sharing, 400
disconnecting from server, 239–240. *See also* connecting to server
disk
independent
platform testing with, 130
snapshots and, 144
types of, 143
nonpersistent
description of, 143
platform testing with, 130
using, 143–144
persistent
description of, 143, 378
platform testing with, 130
physical
installing operating system from setting up, 361
requirements for, 13–14
virtual
creating, 360
pre-allocating, 13, 364
size of, 363
storing, 108–109
disk size
compression of files and, 140
configuring for virtual machine, 107–108
Display tab (console preferences), 71–72
displaying state of registered virtual machines, 171–173
documentation, 127
downloading installation package, 19, 24–25
drag and drop functionality, 69

E

Edit System Variable screen, 212
Edit virtual machine settings (Command section), 72
enabling. *See also* disabling
automatic startup and shutdown of guest, 81–82, 83
File and Printer sharing, 401
MUI and console ports in firewall, 54–55
periodic logging, 84
End User License Agreement, 98
enforcing standards, 133
enumeration
definition of, 160
VmCOM API, 177–180
environment. *See also* development environment
defining target, 351–353
rollout of new, 127
testing, isolating, 91
X, restarting, 115
Environment Variables button, 210
Environment Variables screen, 211

error codes
Programming API, 225–226, 417–421
VmCOM API, 409–411
VmPerl API, 413–415
error handling, 302
`/etc/services` **file, 51**
Ethernet card, virtual, 75
evaluation, built-in functions for, 225–226
event types and callback function, 230
examples
of functions, 230
integration scenarios, 209
`VixCopyToGuest.cpp` file, 271–275
`VixRegisterVM.cpp` file, 270–271
`VmCollection` object, 171–173
`VmCtl object`, 168–171
`VmQuestion` object, 174–177
`VmServerCtl` object, 163–164
`VMware::VmPerl::Question` module, 200–204
`VMware::VmPerl::Server` module, 188–191
`VMware::VmPerl::VM` module, 195–199
Exceptions tab (Windows Firewall dialog box), 401–402
`Execute()` **method, 333, 334–335**
executing program on guest machine, 263–265
exit, in Perl, 191
expanding virtual disk, 13

F

FAT16-based file system, 105
FAT32-based file system, 105
file. ***See also*** `ccnet.config` **file**
answer
creating, 369–372
storing, 121
batch, 281, 342–347
`.cab`, 117, 368
copying
from guest machine to host machine, 266–267
from host machine to guest machine, 265–266
to virtual machine, 271–275
`deploy.cab`, 116, 117
`/etc/services`, 51
header, pointing project to, 216–218
library, pointing project to, 218–220
`.log`, 101
`nvram`, 101
`.vbs`, 160
`.vmdk`, 101
`.vmdkREDO`, 101
`.vmsn`, 101
`.vmx`, 93, 101
`.vmx.sav`, 101
`.wsf`, 160
XML, 333, 347–348
File and Printer sharing, 305, 401
File Download screen, 39
file extensions, 101
File® New® Project, 212
File® Source Control, 286, 319–320
Filter trigger (CCNet), 289
finding running virtual machine, 240–243
firewall, enabling MUI and console ports in, 54–55
flexibility
of image names, 139
of VMware products, 8
floppy drive, virtual, 74, 365
Folder Options dialog box, View tab, 401
Force Build button (CCNet), 295
forcing
build, 388–391
`vmExecutionState_Stuck` state, 174–177
`VM_EXECUTION_STATE_STUCK` state, 200–204
freeing string buffer, 235–236
freezing Windows guest, 116–123
full screen view, 72
functions (Programming API)
built-in, for evaluation, 225–226
error code and, 225–226
general use, 230–237
host machine operations, 237–244
job handle operations, 268–269
referencing with `include` statement, 215
virtual machine operations
`VixVM_CopyFileFromGuestToHost` function, 266–267
`VixVM_CopyFileFromHostToGuest` function, 265–266
`VixVM_CreateSnapshot` function, 251–253
`VixVM_Delete` function, 250–251
`VixVM_GetNumRootSnapshots` function, 253–254
`VixVM_GetRootSnapshot` function, 253–254
`VixVM_InstallTools` function, 258–259
`VixVM_LoginInGuest` function, 262–263
`VixVM_Open` function, 245–246
`VixVM_PowerOff` function, 247–248
`VixVM_PowerOn` function, 246–247
`VixVM_RemoveSnapshot` function, 255–256
`VixVM_Reset` function, 248–249
`VixVM_RevertToSnapshot` function, 256–258
`VixVM_RunProgramInGuest` function, 263–265
`VixVM_Suspend` function, 248–249
`VixVM_UpgradeVirtualHardware` function, 261–262
`VixVM_WaitForToolsInGuest` function, 259–261

G

General section (Options tab), 75–76
General tab (host settings), 62–63
get, definition of, 160
`GetProperties()` **function, 224**
`get_tools_last_active()` **method (VmPerl API), 195**
goals, defining, 353–354
graphics, virtual, 74
guest. ***See also*** **base image**
copying file from, to host machine, 266–267
copying file to, from host machine, 265–266
description of, 6
enabling automatic startup and shutdown of, 81–82, 83
executing program on, 263–265
freezing with Sysprep, 116–123

guest (continued)
mouse, keyboard, and, 68–69
per processor, recommended standard for, 12
GZip (GNU zip) compression utility, 25, 140

H

handle properties (Programming API), 222–224
handle types (Programming API), 222
handles (Programming API)
description of, 220–222
representing root snapshot, getting, 254–255
hard disk, requirements for, 13–14
hardware
default, and settings for operating system, 98–99
for remote client, 17–18
virtual machine, specifications for, 73–75
hardware icons, 112
Hardware tab (Virtual Machine Settings), 109–110, 148
header file, pointing project to, 216–218
Help® Enter Serial Number, 32
Home tab (server console), 61–62, 355
host handle, 222
host hardware requirements, 11–14
host machine
copying file from guest machine to, 266–267
copying file to guest machine from, 265–266
description of, 6
host machine operations (Programming API), 237–244
host operating system
description of, 6, 24
requirements for
Linux, 15–17
overview of, 14
Windows, 15
host settings (server console)
Connection tab, 66, 67
Devices tab, 66
General tab, 62–63
Memory tab, 63–64
Priority tab, 64–65
Host Virtual Adapters tab (Virtual Network Editor), 91
Host Virtual Network Mapping tab (Virtual Network Editor), 87–90
Host® Virtual Network Settings, 84
hosted architecture, 3–5
host-only networking, 30
hot key, default, changing, 70
Hot Keys tab (console preferences), 70–71
HTTP session time out, 36
hypervisor architecture, 3–5

I

icons
CCNet, 291
CCNet Tray, 388
CD-ROM, 148
hardware, 112
plus, 320
VMware Tools, 116, 376
IDE drive, virtual, 74
identifying
type of handle provided as parameter, 233–234
type of property stored in handle, 232–233
image. *See* base image; development image; ISO image; testing image
imaging workstation, 127
`include` **statement, 215**
independent disk
platform testing with, 130
snapshots and, 144
types of, 143
`Infotype` **constant, 205–206**
Input grabbed state, 65
Input tab (console preferences), 68–69
Input ungrabbed state, 65
installation
of API, 42
of CCNet Tray application, 291, 382–383
of CruiseControl.NET, 285, 381–382
host hardware requirements for, 11–14
host operating system requirements for, 14–17
of Internet services daemon, 50–51
on Linux, 24–34
of operating system onto base image, 109–113
overview of, 34
of remote client
on Linux, 43–48
on Windows, 38–42
remote client requirements for, 17–18
troubleshooting, 48–55
VMware products and, 18
of VMware Server Management Interface on Linux, 34–38
of VMware Tools
on Linux guests, 113–116
`VixVM_InstallTools` function, 258–259
on Windows guests, 113
on Windows, 19–24
of `xinetd`, 50–51
integration. *See also* CruiseControl.NET; Visual Studio Team System
configuring external script for, 394
continuous, 278, 349
creating custom assembly for, 330–333
Linux and, 349
overview of, 277
Perl script for, 394–402
source control and, 353
Intel
PAE standard and, 13
processor requirements, 12
Interacting permission, 93
Internet Explorer-Security Warning screen, 40
Internet services daemon, installing, 50–51
Interval trigger (CCNet), 289, 388
Inventory pane (server console), 58–59
IP address
grabbing, 398–399
of guest, retrieving, 375–377
of virtual machine, retrieving, 305

ISO image
creating, 145–146
description of, 144
installing operating system from, 111
library, creating, 144–148
using, 146–148
isolating testing environment, 91
`ISupportErrorInfo` **interface, 409–410**
iterating list of registered virtual machines, 171–173
iteration labeller, 385

J

job handle
asynchronous operations and, 227–230
description of, 222
properties, 223–224
job handle operations (Programming API), 268–269
Just power off (Snapshots section), 77

K

kernel
compiling modules for, 48–50
installing modules into, 27–28
upgrading, 34
keyboard
focus of, 69
virtual, 75

L

labeler, configuring, 385–386
labeller, default, 385
launching VMware Server Console manually, 52–53
`libdb.so.3`**, troubleshooting missing, 51–52**
library
of base images, 6, 137–140
of ISO images, creating, 144–148
Perl, 184
placing in team project, 342–347
`ThoughtWorks.CruiseControl.MSBuild.XMLLogger` library, 392
library file, pointing project to, 218–220
License Agreement screen
Setup Manager Wizard, 120, 370
VMware Server, 20
licensing issues, 98, 123
Linux
creating ISO files in, 147
host operating system requirements, 15–17
platform testing, 129
remote client
installing on, 43–48
requirements for, 18
user access, configuring on, 95–96
virtual networking, configuring on, 85
VMware Server, installing on, 24–34
VMware Server integration under, 349
VMware Server Management Interface, installing, 34–38
VMware Tools, installing on, 113–116
VMware Tools scripting in, 279
local system account, running virtual machine under, 103
localhost server, as default, 23
location
for base image, choosing, 99
for storing virtual disk, choosing, 108–109
`.log` **file, 101**
Log virtual machine progress periodically option (Advanced section), 84
logging in via code, 262–263
LSI adapter, 105

M

main pane (server console), 59–61
management interface. *See* VMware Server Management Interface, installing; web management interface
masking error code, 225–226
memory
requirements for, 12
virtual, 74
for virtual machine, 104–105, 360
Memory for the Virtual Machine screen (New Virtual Machine Wizard), 104, 360
memory page trimming, disabling, 84
Memory tab (host settings), 63–64
methods
definition of, 160
`VmCtl` object, 166–168
`VmServerCtl` object, 162–163
`VMware::VmPerl::ConnectParams` module, 186
`VMware::VmPerl::Question` module, 199
`VMware::VmPerl::Server` module, 187–188
`VMware::VmPerl::VM` module, 191–195
Microsoft .NET Framework, 283, 297
mode
debugging, 84
quick switch, 72
modifier (~), 35
modules
compiling, 48–50
installing into kernel, 27–28
`mount` **command, 114**
mouse
focus of, 68–69
virtual, 75
moving virtual machine from one server to another, 8
MSBuild, configuring, 296–299, 391–393
multiplatform testing, 9
My Computer® Properties® Advanced, 188
My Computer® Tools® Folder Options, 400

N

Name and Organization screen (Setup Manager Wizard), 371
Name the Virtual Machine screen (New Virtual Machine Wizard), 100, 358
naming. *See also* renaming
base images, 99, 137–138
production images, 140–141
projects, 354–355

naming (continued)
test images, 303–304
virtual machines, 138
NAT (Network Address Translation) networking option, 29–30
NAT network, physical IP address and, 89
NAT Settings window, 90
NAT tab (Virtual Network Editor), 92
.NET Framework v2.0, 283, 297
Netbios timeout settings (Host Virtual Network Mapping tab), 90
Network Address Translation (NAT) networking option, 29–30
Network Connections dialog box, 402
network controller, requirements for, 14
Network Type screen (New Virtual Machine Wizard), 105, 361
networking
bridged, 28–29
host-only, 30
NAT, 29–30
virtual
Automatic Bridging tab, 86
Host Virtual Network Mapping tab, 87–90
overview of, 84–85, 360–361
Summary tab, 85
networking configuration for Linux, 28–30
New or Existing Answer File screen (Setup Manager Wizard), 119, 369
New Project dialog box, 213
New Team Build Type Creation wizard (Visual Studio Team System)
Select and order solutions to build screen, 324
Select build location screen, 326
Select build options screen, 327
Select configurations to build screen, 325
Welcome screen, 323
New Team Project wizard (Visual Studio Team System)
description of, 315
Specify the Settings for the Project Portal screen, 317
Specify the Source Control Settings screen, 317
Specify the Team Project Settings screen, 316
New Virtual Machine option (Home tab), 61
New Virtual Machine Wizard
Allocate all disk space now option, 363
Memory for the Virtual Machine screen, 104, 360
Name the Virtual Machine screen, 100, 358
Network Type screen, 105, 361
overview of, 98
Processor Configuration screen, 103–104, 359
Select a Disk screen, 106, 362
Select a Disk Type screen, 107, 362
Select a Guest Operating System screen, 100, 357
Select I/O Adapter Types screen, 106, 361
Select the Appropriate Configuration screen, 99, 356
Set Access Rights screen, 102, 358
Specify Disk Capacity screen, 107–108
Specify Disk File screen, 108–109, 364
Startup/Shutdown options, 357–358
Startup/Shutdown Options screen, 102–103, 359
Welcome screen, 356
902 port, verifying availability of, 50
nonpersistent, configuring testing images as, 378–380
nonpersistent disk
description of, 143
platform testing with, 130
using, 143–144
Notes area (main pane), 60
`nvram` **file, 101**

O

object
definition of, 159
temporary, creating, 236
object-oriented programming (OOP), 159–160
Open dialog box, 282, 377
Open Existing Virtual Machine option (Home tab), 61
Open Virtual Machine screen, 142, 367
opening. *See also* starting
`.cab` file, 368
`ccnet.config` file, 288, 384
New Virtual Machine Wizard, 98
virtual machine, 245–246
Virtual Network Settings window, 84
Visual Studio IDE, 312
web management interface, 37
operating system. *See also* Linux; Windows
abbreviations for, 139
default hardware items and settings for, 98–99
host, 6, 14–17, 24
installing onto base image, 109–113
operational status, determining, 204
Options tab
Virtual Machine Settings window
Advanced section, 82–84
General section, 75–76
Permissions section, 78, 79
Power section, 76–77
sections in, 141–142
Snapshots section, 77–78
Startup/Shutdown section, 78–82
web management interface, 81
organizing library of base images, 137–140
`$OSNAME` **(**`$^O`**) scalar, 184**
overkill, 354

P

PAE (Physical Address Extension), 13
parallel (LPT) port, virtual, 75
parsing list of virtual machines, 302–304
patch level, 139
PCI slot, virtual, 74
Pending Changes tab (Visual Studio IDE), 320, 321, 347
Pending Checkins page (Visual SourceSafe), 287
performance and ISO images, 144
periodic logging, enabling, 84
Perl
cross-platform scripts, 183–185
description of, 153, 183
non-zero returns and, 413
regular expressions, 303–304
VMware Server integration scripts, 394–402
Perl interpreter, 154, 183

permissions, 93, 96
Permissions section (Options tab), 78, 79
persistent disk
description of, 143, 378
platform testing with, 130
Physical Address Extension (PAE), 13
physical disk
installing operating system from, 111
setting up, 361
planning base image, 132–133
platform. *See also* platform testing
multiple, available on single machine, 7
64-bit, 353
supported, 153–154, 352
target environment and supported, 352
platform availability for support analysis, 131
platform testing
Linux, 129
multiplatform testing, 9
overview of, 128
with persistent and nonpersistent disks, 130
with snapshots, 129
plus icon, 320
polling
guest machine to determine if VMware Tools is running, 259–261
for job completion, 227–228, 269
port
default, 31
in firewall, enabling, 54–55
verifying availability of 902, 50
virtual, types of, 75
Port Forwarding button (Host Virtual Network Mapping tab), 90
post-release maintenance, virtual machines for, 131
Power section (Options tab), 76–77
power transition, behavior during, 204–205
powering off virtual machine, 247–248
powering on
first available machine, 304
function for, 246–247, 271–275
virtual machine, 111, 169–171
pre-allocated virtual disk, 13, 364
preparing base images for use, 140–141
Priority tab (host settings), 64–65
process control for target environment, 353
processor
product activation and, 127
requirements for, 12
virtual, 73
Processor Configuration screen (New Virtual Machine Wizard), 103–104, 359
product activation and processor, 127
Programming API (Vix)
asynchronous operations and job handle, 226–230
bug in v1.0 of, 275
compiling against, with Visual C++, 209–220
description of, 154, 155, 156, 209
error codes, 225–226, 417–421
examples, 269–275
functionality of, 153
functions, 230–237
handle properties, 222–224
handle types, 222
handles, 220–222
host machine operations, 237–244
job handle operations, 268–269
release notes for version 1.0, 209
as supported scripting method, 207
as thread safe, 227
virtual machine operations
`VixVM_CopyFileFromGuestToHost` function, 266–267
`VixVM_CopyFileFromHostToGuest` function, 265–266
`VixVM_CreateSnapshot` function, 251–253
`VixVM_Delete` function, 250–251
`VixVM_GetNumRootSnapshots` function, 253–254
`VixVM_GetRootSnapshot` function, 253–254
`VixVM_InstallTools` function, 258–259
`VixVM_LoginInGuest` function, 262–263
`VixVM_Open` function, 245–246
`VixVM_PowerOff` function, 247–248
`VixVM_PowerOn` function, 246–247
`VixVM_RemoveSnapshot` function, 255–256
`VixVM_Reset` function, 248–249
`VixVM_RevertToSnapshot` function, 256–258
`VixVM_RunProgramInGuest` function, 263–265
`VixVM_Suspend` function, 248–249
`VixVM_UpgradeVirtualHardware` function, 261–262
`VixVM_WaitForToolsInGuest` function, 259–261
`vmrun` utility and, 157
Programs® Settings® Session® Startup Programs, 116
Project dialog box (CCNet Tray), 390
Project screen (CCNet), 293
properties
definition of, 160
handle, 222–224
Perl and, 185
`VmCollection` object, 171
`VmConnectParams` object, 161–162
`VmCtl` object, 164–166
`VmQuestion` object, 174
`VmServerCtl` object, 162
Property Pages screen
C/C++ node, 216, 217
Configuration Properties node, 216
Linker node, Input option, 218–219
ProVMwareSampleApp screen (Visual Studio Team System), 344
pushing project payload to desktop of virtual machine, 305–306

Q

quick switch mode, 72

R

read and execute permissions, 93
read permissions, 93
read, write, and execute permissions, 93, 96
recycling testing image, 143–144

reference count for handle, 221, 231
References tab (Solution Explorer, VSTS), 332
referencing
functions with `include` statement, 215
objects with handles, 220
registering virtual machine
remotely, 270–271
with server, 163–164, 188–191, 243
regular expressions (Perl), 303–304
Release screen (Visual Studio Team System), 343
Release To Manufacture (RTM) version, 128, 139
remote client
definition of, 17, 38
installing
on Linux, 43–48
on Windows, 38–42
requirements for, 17–18
remote connection for server console, configuring, 31
removing snapshot, 255–256
renaming
virtual development machine configuration file, 365–367
virtual disk file, 108, 141, 363
Reseal method (Sysprep), 122, 374
resetting virtual machine, 248–249
restarting X environment, 115
retrieving
IP address
of guest, 375–377
of virtual machine, 305
list of registered virtual machines, 300–302
property from handle, 224, 231–232
string description of error passed in, 234–235
Revert to snapshot (Snapshots section), 77
reverting
to snapshot, 129
to state present in snapshot, 256–258
rollout of new environment, 127
root project node, configuring, 384, 385
root snapshot, determining number of, 253–254
RTM (Release To Manufacture) version, 128, 139
Run with debugging information option (Advanced section), 84
running
base image with VMware Player, 135
power scripts, 76
running virtual machine, finding, 240–243

S

saving script, 188
scalar, 184
Schedule trigger (CCNet), 289, 388
script, saving, 188
scripting, 151. *See also* VMware Tools scripting
scripting APIs
depreciation of, 179, 180
future of, 207
supported platforms, 153–154
Scripts tab (VMware Tools Properties screen), 281, 282, 376, 377
SCSI device, virtual, 74
security identifier (SID), 116, 142
Security Level Configuration screen, 54, 55
Security tab (Virtual Machine Properties), 94
security, user access level options, 92–93
Select a Disk screen (New Virtual Machine Wizard), 106, 362
Select a Disk Type screen (New Virtual Machine Wizard), 107, 362
Select a Guest Operating System screen (New Virtual Machine Wizard), 100, 357
Select and order solutions to build screen (New Team Build Type Creation wizard), 324
Select build location screen (New Team Build Type Creation wizard), 326
Select build options screen (New Team Build Type Creation wizard), 327
Select configurations to build screen (New Team Build Type Creation wizard), 325
Select I/O Adapter Types screen (New Virtual Machine Wizard), 106, 361
Select the Appropriate Configuration screen (New Virtual Machine Wizard), 99, 356
serial (COM) port, virtual, 75
serial number, 32
server
connecting to
function example, 271–275
Perl script, 396
`VixHost_Connect` function and, 237–239
`VmCtl` object and, 169–171
disconnecting from, 239–240
registering virtual machine with, 163–164, 188–191, 243
server console
Connect to Host screen, 46, 47
console preferences, 62, 66–72
Home tab, 61–62, 355
host settings, 62–66
installation wizard, 40
Inventory pane, 58–59
launching manually, 52–53
Linux, 33
main pane, 59–61
screen, 57–58
Switch Host screen, 41
updating, 68
Windows, 23
`$server®get_last_error()` **method, 413**
Set Access Rights screen (New Virtual Machine Wizard), 102, 358
set, definition of, 160
`setguid` **sticky bit, 95**
Setup Manager Wizard (Sysprep)
answer file
saving, 371, 372
storing, 121
categories, 121
Completing Setup Manager screen, 371, 372
License Agreement screen, 120, 370
Name and Organization screen, 371
New or Existing Answer File screen, 119, 369
Type of Setup screen, 119, 370

Setup Type screen, 20, 21
shebang (#!), 153, 184
shortcuts, ~directory, 24
`shutdown` **command, 115**
SID (security identifier), 116, 142
simple file sharing, 400
single-thread operations, 236–237
64-bit platform, 353
snapshot
as alternative to nonpersistent disk, 144
creating, 251–252
getting handle that represents, 254–255
platform testing with, 129, 130
removing, 255–256
reverting virtual machine to state present in, 256–258
root, determining number of, 253–254
use of, 134
snapshot errors, Programming API, 420–421
snapshot handle
description of, 222
properties, 224
Snapshots section (Options tab), 77–78
software
antivirus, 134
for virtual development machine, setting up for deployment, 367–374
software development, virtual machines for, 126–127
software testing, virtual machines for, 128–130
software virtualization, 2
Solution Explorer (Visual Studio Team System), References tab, 332
sound adapter, virtual, 75
source control
adding, 289–291
adding developmental project to, 285–288
configuring, 387
target environment, 353
Source Control Explorer (Visual Studio Team System), 322, 346, 348
Specify Disk Capacity screen (New Virtual Machine Wizard), 107–108
Specify Disk File screen (New Virtual Machine Wizard), 108–109, 364
Specify the Settings for the Project Portal screen (New Team Project wizard), 317
Specify the Source Control Settings screen (New Team Project wizard), 317
Specify the Team Project Settings screen (New Team Project wizard), 316
Split disk into 2 GB files option (New Virtual Machine Wizard), 363
SSL certificate, 32, 37
standard errors, Programming API, 418–420
standards, enforcing, 133
Start menu, 23
starting. ***See also*** **opening**
CCNet application, 292
virtual machine, 195–199
Startup/Shutdown options (New Virtual Machine Wizard), 102–103, 357–358
Startup/Shutdown Options screen (New Virtual Machine Wizard), 102–103, 359
Startup/Shutdown section (Options tab), 78–82
state, definition of, 159
state file location, configuring, 384–385
storing
answer file, 121
virtual disk, 108–109
string buffer, freeing, 235–236
`su -` **(switch user) command, 27**
subnet, placing development machines on different, 132
Subnet window, 88
Summary tab (Virtual Network Editor), 85
support analysis, 131
supported platform, 352
suspending virtual machine, 249–250
Switch Host option (Home tab), 62
switch user (`su -`) command, 27
switching view, 71–72
synchronous method, 167
synchronous operation, 226
Sysprep (System Preparation Tool) utility
blank fields and, 371
description of, 367
folder, creating, 368–369
freezing Windows guest with, 116–123
Reseal button, 374
Setup Manager, 119–121, 369–371
tool screen, 373–374
versions of, 368
System Properties screen, 210, 211
System Startup and Shutdown Defaults screen, 83
System Tools menu (Linux), 33

T

tabs (main pane), 60
tar (tape archive), 25
target environment
defining, 351
platform, 352
source and process control, 353
tools, 352
`Task` **class, 333**
tasks to automate, 152
team
collaborating with on base image, 133
common environment for, 126–127
team build
configuring, 322–328
testing, 328–330
Team Explorer pane (Visual Studio Team System), 314, 319
Team Foundation Server, 319, 330
team project
adding development project to, 318–322
creating, 312–318
placing class library and batch file in, 342–347
teaming errors, Programming API, 421
temporary object, creating, 236
test bed, 8, 9
testing. ***See also*** **platform testing**
development image, 134–135
team build, 328–330
testing environment, isolating, 91

testing image
configuring as nonpersistent, 378–380
description of, 138
naming, 303–304
recycling, 143–144
setting up, 374–377
`test-winxppro-sp2` **file, 302–304**
text console, installing VMware Tools and, 114
`ThoughtWorks.CruiseControl.MSBuild.XMLLogger` **library, 392**
tilde (~)
directory shortcut, 24
modifier, 35
Title and Status area (main pane), 60
`/tmp` **folder, 115**
tools, for target environment, 352
tools screen (Visual Studio Team System), 345
Tools® Connect to Team Foundation Server, 313
Tools® Folder Options, 400
`ToolsLastActive` **property (**`VmCtl` **object), 166**
trigger, adding, 288–289
trigger block, configuring, 387–388
troubleshooting installation
of guest operating system, 112–113
of VMware Server, 48–55
Type of Setup screen (Setup Manager Wizard), 119, 370

U

undefined, in Perl, 191
uninstalling previous VMware product, 18
unregistering virtual machine, 244
updating
after deployment, 133–134
console, 68
upgrading
kernel, 34
virtual hardware, 261–262
USB port, virtual, 75
user context, running virtual machine under, 103
user rights to virtual machine
categories of, 80
configuring, 93–96
levels, 92–93
`using` **declarations, 333**
utilities
GZip (GNU zip) compression, 25, 140
Sysprep (System Preparation Tool), 116–123, 367–374
`vmrun` utility, 157
`vmware-cmd` utility, 156
utilization of hardware, 8

V

VB.NET, scripting VMware Tools with, 283–284
`.vbs` **file, 160**
VBScript, scripting VMware Tools with, 279–282
verifying
connection to server, 187, 192
port 902 availability, 50
`VmExecutionState`, 178
view, switching, 71–72
View tab (Folder Options dialog box), 401
View® Other Windows® Pending Changes, 321
View® Pending Checkins, 287
virtual BIOS, 74
virtual chip set, 74
virtual development machine
deleting floppy drive, 365
renaming configuration file, 365–367
setting up, 355–364
setting up software for deployment, 367–374
virtual disk
creating, 360
pre-allocating, 13, 364
size of, 363
storing, 108–109
virtual disk file, renaming, 108, 141, 363
virtual Ethernet card, 75
virtual floppy drive, 74
virtual graphics, 74
virtual hardware, upgrading, 261–262
virtual IDE drive, 74
virtual keyboard, 75
virtual machine
connecting to, powering on, and copying file to, 271–275
creating, 98–109
deleting, 250–251
description of, 101
installing operating system onto, 109–113
iterating list of registered, and displaying state of, 171–173
naming, 138
opening, 245–246
for post-release maintenance, 131
powering off, 247–248
powering on, 111, 169–171, 246–247
registering
remotely, 270–271
with server, 163–164, 188–191
resetting, 248–249
for software development, 126–127
for software testing, 128–130
starting, 195–199
suspending, 249–250
unregistering, 244
user rights to, 80
virtual machine handle
description of, 222
properties, 223
Virtual Machine Properties, Security tab, 94
virtual machine security, 92–93
Virtual Machine Settings window
components of, 72–73
Floppy node, 365
Hardware tab, 109–110, 148
Options tab, 75–84, 141–142
Virtual Machine Startup and Shutdown screen, 82
virtual memory, 74
virtual mouse, 75
Virtual Network Editor
Automatic Bridging tab, 86
DHCP tab, 92
Host Virtual Adapters tab, 91

Host Virtual Network Mapping tab, 87–90
NAT tab, 92
Summary tab, 85
Virtual Network Settings window, opening, 84
virtual networking, 84–85, 360–361
virtual parallel (LPT) port, 75
virtual PCI slot, 74
virtual processor, 73
virtual SCSI device, 74
virtual serial (COM) port, 75
virtual sound adapter, 75
virtual USB port, 75
virtualization
of development server, 9
host machine and guest, 6
hosted versus hypervisor architecture, 3–5
software, 2
virtualization layer, 2
Visual C++, compiling against Vix with
`include` statement and, 215
overview of, 209–210
pointing project
to header files, 216–218
to libraries used, 218–220
setting up project, 210–215
Visual SourceSafe
Add to SourceSafe dialog box, 286–287
installing, 285
logging into, 286
Pending Checkins page, 287
Visual Studio IDE
opening, 312
Pending Changes tab, 320, 321, 347
Visual Studio Team System (VSTS). *See also* build integration, configuring
adding new assembly to team build XML file, 347–349
build hooks, 330
coding build task, 333–337
continuous integration and, 278
copying binaries to virtual machine, 337–338
custom assembly, creating for integration, 330–333
description of, 151, 312
New Team Build Type Creation wizard, 323–327
New Team Project wizard, 315–318
placing class library and batch file in team project, 342–347
team build
configuring, 322–328
testing, 328–330
team project
adding development project to, 318–322
creating, 312–318, 319
`VMIntegration` class, 338–342
VMware Server integration, 330–349
Vix. *See* Programming API (Vix)
`Vix_CreateTemporaryObject()` **function, 236**
`VixError` **object, 417–418**
`VIX_ERROR_CODE()` **function, 225**
`VIX_FAILED()` **function, 226**
`Vix_FreeBuffer()` **function, 235–236**
`Vix_GetErrorText()` **function, 234–235**
`Vix_GetHandleType()` **function, 233–234**
`Vix_GetProperties()` **function, 231–232**
`Vix_GetPropertyType()` **function, 232–233**
`VixHost_Connect()` **function, 237–239**
`VixHost_Disconnect()` **function, 239–240**
`VixHost_FindItems()` **function, 240–243**
`VixHost_RegisterVM()` **function, 243**
`VixHost_UnregisterVM()` **function, 244**
`VixJob_CheckCompletion()` **function, 227, 269**
`VixJob_GetError()` **function, 268–269**
`VixJob_Wait()` **function, 228–230, 268**
`VIX_OK` **error code, 225**
`VIX_PROPERTY_NONE` **property, 232**
`VIX_PROPERTY_VM_TOOLS_STATE` **property, 275**
`Vix_PumpEvents()` **function, 236–237**
`Vix_ReleaseHandle()` **function, 231**
`VIX_SUCCEEDED()` **function, 226**
`VixVM_CopyFileFromGuestToHost()` **function, 266–267**
`VixVM_CopyFileFromHostToGuest()` **function, 265–266**
`VixVM_CreateSnapshot()` **function, 251–253**
`VixVM_Delete()` **function, 250–251**
`VixVM_GetNumRootSnapshots()` **function, 253–254**
`VixVM_GetRootSnapshot()` **function, 254–255**
`VixVM_InstallTools()` **function, 258–259**
`VixVM_LoginInGuest()` **function, 262–263**
`VixVM_Open()` **function, 245–246**
`VixVM_PowerOff()` **function, 247–248**
`VixVM_PowerOn()` **function, 246–247**
`VixVM_RemoveSnapshot()` **function, 255–256**
`VixVM_Reset()` **function, 248–249**
`VixVM_RevertToSnapshot()` **function, 256–258**
`VixVM_RunProgramInGuest()` **function, 263–265**
`VixVM_Suspend()` **function, 249–250**
`VixVM_UpgradeVirtualHardware()` **function, 261–262**
`VixVM_WaitForToolsInGuest()` **function, 259–261**
`VmCollection` **object, 171–173**
VmCOM API
description of, 153–154, 155, 156
enumerations, 177–180
error codes, 409–411
future of, 207
objects, 161
`VmCollection` object, 171–173
`VmConnectParams` object, 161–162
`VmCtl` object, 164–171
`VmQuestion` object, 173–177
`VmServerCtl` object, 162–164
`VmConnectParams` **object, 161–162**
`VmCtl` **object**
example, 168–171
methods, 166–168
`.vmdk` **file, 101**
`.vmdkREDO` **file, 101**
`VM_EXECUTION_STATE` **constant, 204**
`VmExecutionState` **enumeration, 177–178**
`VmExecutionState`**, verifying, 178**
`vmExecutionState_Stuck` **state, 173, 174–177**
`VM_EXECUTION_STATE_STUCK` **state, 200–204**
`$vm®get_last_error()` **method, 413–414**
`VMIntegration` **class, 337, 338–342**

VMnet0, 85, 86, 87
VMnet1, 85, 87, 89
VMnet8, 85, 89
VmPerl API
constants, 204–207
description of, 153, 154–156
error codes, 413–415
future of, 207
modules, 185
`VMware::VmPerl::ConnectParams` module, 185–187
`VMware::VmPerl::Question` module, 199–204
`VMware::VmPerl::Server` module, 187–191
`VMware::VmPerl::VM` module, 191–199
`VmPlatform` **enumeration, 179–180**
`VM_POWEROP_MODE` **constant, 204–205**
`VmPowerOpMode` **enumeration, 178**
`VM_PRODINFO_PLATFORM` **constant, 206**
`VM_PRODINFO_PRODUCT` **constant, 206**
`VmProdInfoType` **enumeration, 179**
`VmProduct` **enumeration, 179**
`VmQuestion` **object, 173–177**
`vmrun` **utility, 157**
`VmServerCtl` **object, 162–164**
`vmSetIP.cs` **file, 375–376**
VM® Settings, 72
`.vmsn` **file, 101**
`.vmss` **file, 101**
`vmTestBank.pl`
code, 307–310
file, 404–407
VMware Infrastructure, 157
VMware Player, running base image with, 135
VMware Server
ability to move virtual machines from one server to another, 7–8
Administration guide, 12
advantages of, 6–9
development process and, 8–9
increased utilization of hardware, 8
multiple platforms available on single physical machine, 7
parts of, 11
rapid deployment and, 6–7
VMware Server Console
Connect to Host screen, 46, 47
console preferences, 62, 66–72
Home tab, 61–62, 355
host settings, 62–66
installation wizard, 40
Inventory pane, 58–59
launching manually, 52–53
Linux, 33
main pane, 59–61
screen, 57–58
Switch Host screen, 41
updating, 68
Windows, 23
VMware Server Management Interface, installing
on Linux, 34–38
remote client via
on Linux, 43–48
on Windows, 38–42
VMware Software Development Kit (SDK), 157
VMware Tools
authenticating and logging in using, 262–263
batch files, 281
description of, 69
installing
on guest, 258–259
on Linux guests, 113–116
on Windows guests, 113
polling guest machine to determine if running, 259–261
VMware Tools icon, 116, 376
VMware Tools Properties screen, Scripts tab, 281, 282, 376, 377
VMware Tools scripting
with C# and VB.NET, 283–284
overview of, 279
with VBScript, 279–282
VMware VmPerl Scripting API, 32
`vmware-cmd` **utility, 156**
`vmware-config.pl` **script, 27**
`vmware-install.pl` **script, 25**
`vmware.log` **file, 101**
`vmware-users` **group, creating, 95–96**
`VMware::VmPerl::ConnectParams` **module, 185–187**
`VMware::VmPerl::Question` **module, 199–204**
`VMware::VmPerl::Server` **module, 187–191**
`VMware::VmPerl::VM` **module, 191–199**
`.vmx` **file, 93, 101**
`.vmx.sav` **file, 101**
VSTS (Visual Studio Team System). ***See also*** **build integration, configuring**
adding new assembly to team build XML file, 347–349
build hooks, 330
coding build task, 333–337
continuous integration and, 278
copying binaries to virtual machine, 337–338
custom assembly, creating for integration, 330–333
description of, 151, 312
New Team Build Type Creation wizard, 323–327
New Team Project wizard, 315–318
placing class library and batch file in team project, 342–347
team build
configuring, 322–328
testing, 328–330
team project
adding development project to, 318–322
creating, 312–318, 319
`VMIntegration` class, 338–342
VMware Server integration, 330–349

W

web management interface
description of, 37
installing API via, 42
installing remote client via
on Linux, 43–48
on Windows, 38–42
Startup and Shutdown configuration, 80–82, 83
Web sites
CruiseControl.NET, 285
`ThoughtWorks.CruiseControl.MSBuild.XMLLogger` library, 392
VMware Server Administration guide, 12
WinImage, 111
Wrox, 278
Welcome screen
New Team Build Type Creation wizard, 323
New Virtual Machine Wizard, 356
Visual SourceSafe, 286
VMware Server, 19
Win32 Application Wizard, 214
Windows
configuring user access on, 94–95
freezing guest with Sysprep, 116–123
host operating system requirements, 15
remote client
installing on, 38–42
requirements for, 18
VMware Server, installing on, 19–24
VMware Tools, installing on, 113
Windows Firewall dialog box, Exceptions tab, 401–402
Windows 98 Second Edition, 374
Windows Scripting Host, 160
WinImage program, 111, 145
WinRAR format, 140
Win32 Application Wizard, 213, 214
working directory, 298
Workspace tab (console preferences), 68
workstation, imaging, 127
writing code from within development virtual machine, 9
Wrox Web site, 278
`.wsf` **file, 160**

X

X environment, restarting, 115
`xinetd`**, installing, 50–51**
XML file, 333, 347–348
XML logging, 298, 391–393